WorkLoadWeaver: Realistic Synthetic Query Workloads for Adaptive Database Systems

Keming Li, Nada Lahjouji, Ashwin Gerard Colaco, Shiyuan Zhou, Pratyoy Das, Guangxue Zhang, Sahil Makhijani, Vishal Chakraborty, Sarvesh Pandey, Sharad Mehrotra*
University of California, Irvine
Irvine, CA, USA
{kemingl1,nlahjouj,acolaco,szhou20,pratyoyd,gxzhang7,sahilrm,vchakrab,sarvesh1}@uci.edu,sharad@ics.uci.edu

Abstract

Self-driving data systems require realistic query workloads to train and evaluate adaptive components, but production traces are rarely shareable, and template-based benchmarks miss key dynamics such as temporal variation, session-driven dependencies, refinements, and workload shift. We present WorkLoadWeaver, a benchmark-grounded pipeline that constructs a workload specification from a set of query templates and optional domain assumptions. The specification models personas and temporal profiles, persona-conditioned query transitions with within-template refinements, and event-driven perturbations. WorkLoadWeaver additionally supports varying user participation, ranging from fully automated spec construction to human checkpoints that provide structured summaries and allow edits and constraints.

CCS Concepts

• **Information systems** → **Database performance evaluation**; *Database query processing*; Autonomous database administration; • **Computing methodologies** → *Multi-agent systems.*

Keywords

workload generation, synthetic benchmarks, query workloads, self-driving databases, multi-agent systems, workload specification

ACM Reference Format:
Keming Li, Nada Lahjouji, Ashwin Gerard Colaco, Shiyuan Zhou, Pratyoy Das, Guangxue Zhang, Sahil Makhijani, Vishal Chakraborty, Sarvesh Pandey, Sharad Mehrotra. 2026. WorkLoadWeaver: Realistic Synthetic Query Workloads for Adaptive Database Systems. In *Workshop on Synthetic Data Generation and Management for Building AI Systems (SynthAI '26), May 31-June 05, 2026, Bengaluru, India.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3814574.3816743

1 Introduction

Modern data systems [4, 14–16] are increasingly expected to drive and manage themselves autonomously. They tune their own physical design, adapt their execution strategies, and respond to shifting workloads — all without manual intervention. To do this, they rely on workload traces: ordered sequences of queries that inform the system what to observe, learn, and optimize. However, building and evaluating these self-driving components requires realistic workloads. Without them, it is hard to know whether a system will hold up under the conditions it will face in deployment.

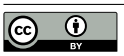
Real-world traces are rarely available outside the organizations that produce them, due to sensitive business logic, proprietary schema details, and private user data. Synthetic benchmarks such as TPC-H and TPC-DS [18, 19] are natural fallbacks. However, they fall short. They offer a fixed set of templates and overlook the dynamics that matter most: temporal bursts and cycles, users refining and drilling down over the course of a session, and distribution shifts triggered by external events. Recent work confirms that they also miss patterns commonly seen in practice, such as query repetition and gradual workload drift [22]. Beyond query patterns, prior work also highlights limitations in how these benchmarks model skew and correlation structure. For example, assuming independence across columns and tables produces unrealistic data distributions, meaning self-driving components are never stress-tested against the skewed, correlated workloads they will face in practice [3].

LLM-based synthesis is a promising path forward, capable of generating richer and more flexible workloads [17]. However, it introduces new risks around grounding and hallucination [23]. What is needed, therefore, is a synthesis approach that combines the structure and reliability of template benchmarks with the flexibility to model real workload dynamics.

Toward this goal, we introduce WorkLoadWeaver, a benchmark-grounded pipeline for constructing workload specifications that can be replayed as query traces. WorkLoadWeaver takes as required input a query template benchmark, including schema and templates, along with optional domain assumptions and human defined rules, and optionally a dataset or data generator to infer data-driven correlations. It produces a structured workload specification with three components: a persona-driven temporal model specifying how different categories of actors generate queries over time; a dependency model capturing query-to-query transitions and within-template refinements; an event and perturbation model that simulates distribution shift and event-triggered querying.

WorkLoadWeaver is realized as a pipeline of sequential agents. The **Persona Agent** constructs the behavioral layer of the workload: it proposes personas, their counts, temporal intensity profiles, per-persona query template transition graphs, and within-template refinement operator distributions that govern how users iteratively adjust parameter values (e.g., tightening a date range or changing a grouping column) across consecutive queries in a session. The **Parameter Dependency Agent** grounds the parametric layer: it

*Sharad Mehrotra is the corresponding author.



This work is licensed under a Creative Commons Attribution 4.0 International License.
SynthAI '26, Bengaluru, India
© 2026 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-2581-4/2026/05
https://doi.org/10.1145/3814574.3816743

proposes value domains, predicate range bands, and attribute correlation structure for template slots, optionally using a dataset or data generator analyzer to derive empirically grounded dependencies; absent such input, the agent makes explicit assumptions about value distributions and selectivity. The **Event and Perturbation Agent** constructs the shift layer: it proposes event occurrence models, trigger mappings from events to query templates, and perturbation rules that simulate shifts in workload distribution.

After each agent stage, WorkLoadWeaver emits a structured **Workload Assumption Summary** listing the generated model components and all assumptions made; the user may accept the output to proceed automatically, edit or add constraints to steer the next stage, or iterate until satisfied, giving the pipeline tunable and transparent human involvement.

Contributions. Designing realistic synthetic workloads raises challenges in grounding LLM-based synthesis, modeling intra-session query dependencies, simulating workload shifts without production traces, and maintaining transparent human oversight. We make the following contributions to address them:

- **Grounded synthesis.** Using LLMs for workload synthesis, without careful grounding, may cause models to hallucinate schema semantics and produce queries that look plausible but are structurally unrealistic. We present a multi-agent pipeline that anchors synthesis in a query-template benchmark, optional domain assumptions, and optionally data-grounded dependency signals, ensuring that generated workloads remain aligned to real schema and data.
- **Context-aware query dependencies.** Realistic workloads are not independent draws: users iterate, refine predicates, and branch their analysis across a session in ways that standard benchmarks do not capture. We propose a workload specification that models persona-driven temporal intensity and query-to-query dependencies, reflecting how analysis naturally evolves within a session.
- **Workload shift evaluation in the absence of traces.** Self-driving systems are deployed in environments where workloads shift due to incidents, releases, and seasonal effects, yet production traces for validating synthetic workloads are rarely available. We introduce an event and perturbation model that simulates such shifts, along with a lightweight evaluation protocol to validate synthetic workloads without requiring production traces.
- **Transparent human oversight.** Automated synthesis pipelines can be opaque, making it hard to trust or steer the output. To remedy this, we introduce human checkpoints with structured workload assumption summaries that let users inspect, edit, and constrain generated artifacts at each stage, keeping the pipeline transparent and controllable throughout.

2 Related Work

Template-based benchmarks. TPC-H and TPC-DS [18, 19] are the standard decision support benchmarks, providing schema, parameterized query templates, and a data generator. They expose a fixed template set with independent, identically distributed query draws and no model of temporal dynamics or user behavior. DSB [3] addresses some limitations by introducing richer skew and correlation structure, and simulates workload drift via parameterized statistical shifts; however, its drift model is defined by mathematical functions rather than semantic drivers such as user intent or external events, and it does not model query-to-query dependencies within sessions. CAB [20] introduces multi-tenancy and arrival time patterns grounded in Snowflake traces, but draws queries heuristically from a TPC-H pool without modeling session structure or refinement behavior. Stitcher [21] takes a step toward trace-grounding by using Bayesian optimization to tune query frequencies and concurrency levels from existing benchmarks to match a target performance profile (CPU, I/O, latency); however, it operates on aggregate metrics only, and does not model operator distributions or session dependencies, and still requires a performance trace as its synthesis target. WorkLoadWeaver differs from all of these by generating a full workload specification that models who issue queries, how queries evolve within sessions, and how the workload shifts under external events, requiring only a query template benchmark as input.

Trace-driven synthesis. RedBench [22] extracts workload patterns directly from production Redshift and Snowflake traces, preserving temporal bursts, structural repeats, and high DML ratios, representing the state of the art in trace fidelity. PBench [23] takes a hybrid approach: given aggregate execution statistics from cloud traces (CPU time, query-plan operator distributions) rather than raw SQL, it formulates workload synthesis as a multi-objective ILP over existing benchmark components, using an LLM augmenter to fill gaps. Both systems are fundamentally reactive and trace-dependent; RedBench requires full production SQL logs, and PBench requires trace statistics, but neither can generate what-if scenarios for new schemas, hypothetical user populations, or unseen event types. WorkLoadWeaver is generative and benchmark-grounded, requiring no production traces and producing a controllable, replayable specification from a query template benchmark alone.

LLM-based workload synthesis. SQLBarber [11] uses LLMs to generate SQL templates from natural language specifications and matches a target cost distribution (cardinality, execution plan cost) via a Bayesian optimizer and self-correction loop, grounded in Redshift and Snowflake execution statistics. SQLStorm [17] uses LLMs to generate large, diverse query sets for stress-testing query optimizers and identifying system crashes, prioritizing structural coverage over workload dynamics. Both systems operate at the level of individual query generation: they produce a set or bag of queries optimized for cost fidelity or structural diversity, with no model of temporal intensity, session evolution, or event-driven shift. WorkLoadWeaver addresses this gap by anchoring LLM agents to a supplied query template benchmark, restricting generation to behavioral and distributional parameters rather than raw SQL, and surfacing all assumptions at human checkpoints for inspection and correction.

Semantic query processing and ML-integrated benchmarks. A related line of work studies semantic query processing, where relational queries invoke expensive UDFs or ML models during execution to enrich tuples with derived concepts. Systems such as EnrichDB [4] and JENNER [5], and specialized query optimizers like Raven [7] and Willump [10], focus on reducing the latency of these costly semantic operators through caching, filtering, and approximation. Benchmarks such as SmartBench [6] and TPCx-AI [1] evaluate these capabilities by embedding inference-heavy or semantically enriched queries into fixed scenarios. In contrast,

WorkLoadWeaver does not optimize the execution of semantic operators themselves, but models when and why such enrichment demands arise, including shifts driven by users, sessions, and external events.

3 Workload Specification Model

A workload specification $\mathcal{W}$ is a structured, generative description from which a deterministic sampler produces a query stream at any scale. $\mathcal{W}$ comprises five coupled models, each answering a distinct question about the workload: who issues queries and when, what sequence of templates they follow, how parameter values are drawn and refined within a session, and what external events perturb query traffic and data state. These are: (1) a grounding layer that fixes schema and template constraints, (2) a persona and arrival model, (3) a template transition model, (4) a parameter instantiation and refinement model, and (5) an event and perturbation model. The models are coupled in that they exchange information at generation time, yet separable in that any model can be simplified or disabled without invalidating the others. All assumptions are surfaced in a *Workload Assumption Summary* at human-in-the-loop checkpoints between agents (Section 4), where users may inspect, correct, and iterate before generation proceeds.

We begin with two definitions used throughout.

Definition (Persona). A *persona* is a category of workload-generating actor, either a human role (analyst, operator, customer) or an automated process (ETL job, monitoring daemon, dashboard refresh). A persona describes a *kind* of actor, not a specific individual; many users may instantiate the same persona.

Definition (Session). A *session* is a bounded sequence of queries issued by one instance of a persona within a contiguous time window, where queries are logically related by a shared task or investigation. A session terminates when the user departs, a configurable inactivity timeout elapses, or the session reaches a maximum length.

3.1 Grounding Layer

The grounding layer defines the hard constraints that every generated query must satisfy. It requires two inputs: a schema S specifying tables, columns, types, constraints, and foreign keys, and a set of parameterized query templates $Q = \{q_1, \ldots, q_m\}$, each a SQL statement with named placeholder slots that are filled with concrete values at generation time.

Validity is formalized per parameter slot. Let $\mathcal{D}_{\text{slot}}$ denote the domain of slot σ as declared in S. The grounding layer maintains a validity function $V_\sigma : \mathbb{T} \to 2^{\mathcal{D}_{\text{slot}}}$ mapping each timestamp to the set of admissible values. For static data, V_σ is constant and extracted once from the schema or an optional dataset. For dynamic data, V_σ evolves as the event model fires (Section 3.5): insertions expand the valid set, deletions shrink it, and updates may invalidate previously admissible values. The generator validates every sampled value against $V_\sigma(t)$ at instantiation time and resamples if necessary.

3.2 Persona and Arrival Model

Let $P = \{p_1, \ldots, p_k\}$ be the set of personas, each assigned a relative weight $w_p \geq 0$ with $\sum_p w_p = 1$. The query arrival rate for persona p at time t is

$$\text{rate}_p(t) \;=\; \Lambda \cdot w_p \cdot \lambda_p(t), \tag{1}$$

where $\Lambda > 0$ is a global scale factor (peak queries per unit time) and $\lambda_p(t) \in [0, 1]$ is the persona's temporal intensity profile, with peak value 1.

Temporal intensity. Database workloads exhibit strong diurnal and weekly cycles [12], and sinusoidal rate functions have been shown to fit real-world arrival processes in service systems empirically [8]. The intensity profile is therefore expressed as a weighted sum of Fourier components across three timescales:

$$\tilde{\lambda}_p(t) \;=\; \sum_{s \,\in\, \{\text{hour, day, month}\}} \alpha_{p,s}\, f_{p,s}(t), \tag{2}$$

where each $f_{p,s}$ is a truncated Fourier series with the natural period of timescale s (24 hours, 7 days, 12 months), and $\alpha_{p,s} \geq 0$ are mixing weights with $\sum_s \alpha_{p,s} = 1$. The Fourier coefficients of each $f_{p,s}$ are fit by least squares to a small number of agent/human-supplied control points, such as "peak at hour 9, value 1.0; overnight near 0.1." We then normalize $\lambda_p(t) = \tilde{\lambda}_p(t) / \max_{t'} \tilde{\lambda}_p(t')$ to ensure $\lambda_p(t) \in [0, 1]$.

Using relative weights rather than absolute user counts decouples workload shape from scale: Λ controls total volume, while $\mathbf{w}$ controls the persona mix, making it straightforward to sweep either independently (Section 5).

3.3 Template Transition Model

For each persona p, the template transition model specifies a start distribution π_p over Q and a row-stochastic transition matrix

$$P_p(q' \mid q), \quad q, q' \in Q, \tag{3}$$

giving the probability that a user following persona p issues a query of template q' immediately after one of template q within the same session. Self-loops $P_p(q \mid q) > 0$ naturally capture repeated execution of the same template, common in monitoring and dashboard workloads, without requiring a separate operator. Persona conditioning is essential: an analyst typically drills down from a summary to a detailed query, whereas an ETL process executes a fixed pipeline; a single global matrix would conflate these incompatible behaviors. When finer temporal resolution is needed, the transition probabilities may be conditioned on a small number of time regimes (e.g., peak, off-peak, maintenance window), yielding per-regime matrices $P_{p,r}(q' \mid q)$.

3.4 Parameter Instantiation and Refinement Model

For each parameter slot σ in each template q, the instantiation model specifies a distribution $\mathcal{F}_\sigma$ from which concrete values are drawn at generation time. The refinement model then governs how parameter values are modified within a session before the next template transition fires. Since the query structure is fixed by the template, all refinement operators act exclusively on parameter values, not on the structure of the query.

Parameter distributions. WorkLoadWeaver supports two modes for constructing $\mathcal{F}_\sigma$. When a dataset or data generator is provided, distributions are fit directly to the data: value ranges, categorical frequencies, and cross-slot correlations are extracted from the actual data, yielding grounded, high-fidelity distributions. When only the schema is available, WorkLoadWeaver infers plausible distributions from schema metadata and column semantics. Column types,

constraints, and names (e.g., `order_date`, `discount`, `quantity`) are used to propose initial ranges and distributional shapes. Rather than restricting choices to a fixed menu of standard families, WorkLoadWeaver may propose custom, semantically meaningful distributions, for example, "a heavy-tailed distribution with most queries falling in the last 90 days but occasional full-history scans." These proposals, together with their natural-language justifications, are surfaced in the Workload Assumption Summary for user review and correction.

Slots within a template may be statistically dependent: for example, the start and end of a date range are positively correlated, and a product category may constrain the set of admissible product identifiers. These dependencies are encoded in a per-template dependency graph $\mathcal{G}_q$, a directed acyclic graph over slots where an edge $\sigma \to \sigma'$ indicates that the distribution of σ' is conditioned on the sampled value of σ. Values are drawn in topological order and validated against $V_\sigma(t)$ after each draw.

Table 1: Refinement operators. All operators act on parameter values within a fixed template structure.

Operator	Slot type	Effect
NoAction	None	Useful for static queries
TightenRange	Numeric / date $[lo, hi]$	$[lo + \delta,\ hi - \delta]$, sampled $\delta > 0$
LoosenRange	Numeric / date $[lo, hi]$	$[lo - \delta,\ hi + \delta]$
ShiftWindow	Date $[lo, hi]$	$[lo + \Delta,\ hi + \Delta]$, sampled Δ
ChangeLimit	Integer limit	Resample from $\mathcal{F}_\sigma$
ChangeGroupBy	Categorical	Resample from $\mathcal{F}_\sigma$

Refinement operators. Within a session, a user may adjust parameter values before the next template transition is launched. This behavior is modeled through a small set of refinement operators, each acting on a specific slot type. At each refinement step, an operator is sampled from a per-(persona, template) distribution $P_{p,q}(\text{op})$, applied to the current parameter state, and the result is validated against the grounding layer. Refinement continues until the transition model is triggered. The supported operators are listed in Table 1.

3.5 Event and Perturbation Model

The event model introduces exogenous real-world processes that affect both query traffic and data state independently of user sessions. Each event type e is characterized by three components: an occurrence model $\lambda_e(t)$, a query trigger, and a data-state effect which we describe below.

• The **occurrence model** $\lambda_e(t)$ is a Fourier-parameterized rate function, specified in the same form as persona intensity profiles (Section 3.2). For example, a nightly batch load peaks at 2 am; a monthly financial close fires on the last business day of each month.

• The **query trigger** $P(q \mid e, p)$ specifies which templates fire, and through which persona, when event e occurs. For instance, a batch data load may trigger a burst of post-load validation queries through the ETL persona, followed by dashboard refresh queries through the analyst persona.

• The **data-state effect** describes how the event mutates the valid-value sets maintained by the grounding layer. Three categories of events are supported. *Arrival events* ingest new data (batch loads, streaming inserts), expanding valid ranges and increasing cardinality. *Mutation events* apply updates or deletes (price changes, order cancellations, schema migrations), which may invalidate previously admissible parameter values and shift query traffic toward templates that reflect the changed state (e.g., a cancellation spike drives a burst of refund-lookup queries). *Shock events* represent external disruptions such as a flash sale or a system outage: they drive a sudden change in arrival rate and template distribution for a bounded time window and may coincide with data mutations (a flash sale generates new orders, expanding the valid order-ID range).

The total observed query rate at time t is the sum of persona-driven and event-triggered traffic:

$$\text{total}(t) \;=\; \Lambda \sum_p w_p\, \lambda_p(t) \;+\; \sum_e \lambda_e(t) \cdot \mathbb{E}[\text{queries per } e]. \quad (4)$$

Because all intensity profiles are Fourier-parameterized, the specification supports controlled perturbation: scaling amplitudes adjusts intensity, rotating phases adjusts timing, and injecting harmonics adjusts burstiness, each at preset severity levels (mild, moderate, severe) with measurable spectral distance.

4 WorkLoadWeaver Architecture

WorkLoadWeaver constructs the workload specification $\mathcal{W}$ described in Section 3 through a three-stage pipeline of sequential LLM agents, followed by a deterministic generation stage. Each agent is instructed to generate its output together with the observations or assumptions that justify it. Figure 1 summarizes the end-to-end pipeline. The required inputs are a schema S and a template set Q, which are solid blue rectangles. Optional inputs, which are dash blue rectangles, include a domain description, a dataset or data generator, and a set of per-agent validation rules/policies $\{\mathcal{R}_1, \mathcal{R}_2, \mathcal{R}_3\}$, which encode user-defined constraints for automatically checking whether each intermediate sub-spec is consistent with the domain requirements before the pipeline proceeds.

The three agents are sequential, because each stage depends directly on the structure produced by the previous one. The Persona Agent first proposes the workload personas and assigns templates to persona-specific usage patterns. The Parameter Dependency Agent then uses this persona-template structure to infer parameter distributions and refinement dependencies for the corresponding template and persona combinations. Finally, the Event Agent uses both the persona structure and the parameter dependency graph to generate event-trigger mappings that remain consistent with the existing specification. Running the agents in parallel would force each to operate without context from the others, risking mutually inconsistent sub-specs. Beyond ordering, the sequential design keeps each agent's prompt focused on a single, well-bounded sub-task, increasing the proportion of task-relevant evidence in the LLM's context window and reducing the risk of hallucination compared to a single monolithic prompt covering the full specification surface [2, 9]. Optionally, the user may supply per-stage validation policies

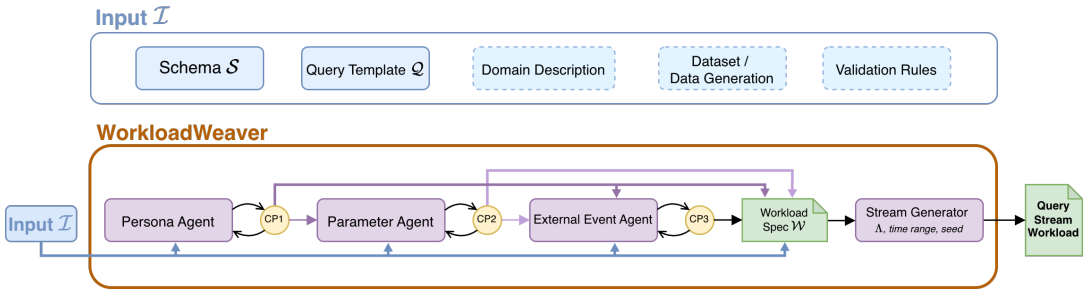


Figure 1: WorkLoadWeaver System Architecture

— sets of predicate rules that automatically check whether each intermediate sub-specification meets domain constraints — reducing the need for manual review.

A checkpoint layer is applied after every agent stage. The system emits a structured *Workload Assumption Summary* for the produced sub-specification, and this summary is then either checked automatically or reviewed by a human operator. When a validation policy $\mathcal{R}_i$ is provided for stage i, each rule $\rho \in \mathcal{R}_i$ is evaluated as a predicate over the produced sub-spec $\hat{\mathcal{W}}_i$:

$$\rho : \hat{\mathcal{W}}_i \rightarrow \{\top, \bot\}, \tag{5}$$

with a diagnostic message emitted on failure. If all rules pass, the pipeline proceeds automatically to the next stage. If no validation policy is provided, or if any rule fails, the summary is surfaced to the operator for human-in-the-loop verification. The operator may accept the result, edit it directly, or provide feedback for regeneration before the next stage proceeds. This design supports a spectrum of usage modes, from manual review at every checkpoint to fully automated execution under user-defined rules.

4.1 Agent 1: Persona Agent

The Persona Agent constructs the behavioral layer of the specification, comprising the arrival, transition, and refinement models of Sections 3.2–3.4. Its inputs are the schema S, the template set Q, and any supplied domain description.

The agent proposes a set of personas, each with a name, role description, and relative weight w_p. For every persona it emits temporal intensity control points at the hourly, daily, and monthly timescales, session characteristics (length distribution, inter-query delay, inactivity timeout), the per-persona template transition matrix $P_p(q' \mid q)$ including self-loop probabilities for repeated template execution, and the within-template refinement operator distribution $P_{p,q}(\text{op})$.

Output: Assumption Summary 1. The summary lists every proposed persona, weight, peak time, transition probability, and refinement operator probability. Each value is tagged as Source (derived from schema constraints or explicit user input) or Assumed (the agent's domain-knowledge estimate). For example: "*Morning Analyst peaks at hour 9 (Assumed: no data to confirm)*" or "*ETL-to-Report transition probability* = 0.6 *(Assumed: estimated, not observed)*." The operator can accept, edit individual values, add constraints, or rerun the agent.

4.2 Agent 2: Parameter Dependency Agent

The Parameter Dependency Agent constructs the parametric layer of the specification, comprising the instantiation and refinement models of Section 3.4. Its inputs are the schema, the template set, and the behavioral sub-spec produced by Agent 1. When a dataset or data generator is provided, the agent analyzes actual data distributions, value ranges, categorical frequencies, and foreign-key cardinalities to produce empirically grounded parameter distributions. When no dataset is available, the agent infers plausible distributions from schema metadata and column semantics, making all assumptions explicit in the Assumption Summary.

For each parameter slot the agent proposes a distribution $\mathcal{F}_\sigma$, which may be a standard family (uniform, Zipfian, normal, categorical) or a custom distribution with a natural-language description when the data warrants it. It also constructs the per-template parameter dependency graph $\mathcal{G}_q$ and proposes predicate range bands governing how refinement operators (e.g., TightenRange) modify the current parameter state.

Output: Assumption Summary 2. Each proposed distribution and dependency is tagged with its provenance. For instance: "`customer_id`: *Zipfian(skew = 1.2, range = [1, 10000]); Source: dataset analysis — top 10% of customers account for 60% of queries*" versus "`s_state`: *categorical, 50 states; Assumed: proportional to population (no query logs).*"

4.3 Agent 3: Event and Perturbation Agent

The Event and Perturbation Agent constructs the event layer of the specification (Section 3.5). Its inputs are the schema, the template set, and the behavioral and parametric sub-specs from the first two agents.

The agent proposes event types (batch loads, mutations, external shocks), occurrence models $\lambda_e(t)$ expressed as Fourier control points, trigger mappings $P(q \mid e, p)$ specifying which templates fire for which persona when an event occurs, and data-state effects describing how events mutate the valid-value sets $V_\sigma(t)$ maintained by the grounding layer.

Output: Assumption Summary 3. Example entries include: "*Nightly batch load at 2 am, duration ~ 1 hr; Assumed: based on common ETL patterns*" and "*Batch load triggers validation queries Q44–Q46 through the ETL persona with P* = 0.8."

4.4 Workload Stream Generator

After all Assumption Summaries have been approved or automatically validated, the three sub-specs are merged into a final workload specification $\mathcal{W}$. The merge stage fits Fourier coefficients to all control points and validates consistency: persona weights sum to 1, all template references exist in Q, transition matrices are row-stochastic, parameter ranges lie within schema constraints, the dependency graph $\mathcal{G}_q$ is acyclic, and event triggers reference valid personas and templates. The validated specification, together with a global scale factor Λ, a time range $[t_{\text{start}}, t_{\text{end}}]$, and a random seed, is passed to the Workload Stream Generator.

The generator advances over discrete time steps. At each step it first processes exogenous events by sampling occurrences from $\lambda_e(t)$, producing event-triggered query candidates via $P(q \mid e, p)$, and updating the valid-value sets $V_\sigma(t)$ according to the data-state effects specified for each event. Next, for each persona p, the generator samples the number of new sessions using a non-homogeneous Poisson process with rate proportional to $\lambda_p(t)$, initializing each session by drawing a starting template from π_p.

For each active session with current template q and parameter state state, the generator constructs a candidate set of next query instances from three sources: template transition candidates from $P_p(q' \mid q)$, refinement candidates obtained by sampling from $P_{p,q}(\text{op})$ and applying the selected operator to the current parameter state, and any event-triggered candidates queued for this persona. Each candidate is scored by its probability under the proposing model. The generator then samples the next query using greedy sampling with temperature: the top k candidates are renormalized and a temperature transform $\tilde{p}_i \propto p_i^{1/T}$ is applied before sampling, where lower T biases toward higher-probability candidates. Parameter values for the selected query depend on whether the session remains on the same template or transitions to a new one. If the selected candidate is a refinement of the current template, its parameter values are produced by applying the sampled refinement operator to the session's current parameter state. If the candidate transitions to a different template, parameter values are produced by evaluating that template's dependency graph $\mathcal{G}_q$ in topological order, sampling each slot from $\mathcal{F}_\sigma$ conditioned on upstream values, and validating against $V_\sigma(t)$. All randomness flows through a seeded pseudo-random number generator, ensuring that the same specification, seed, and time range produce an identical query stream.

5 Preliminary Experiments

We evaluate WorkLoadWeaver along four axes:

- **RQ1 (Structure):** Do WorkLoadWeaver traces exhibit richer session structure than IID baseline methods?
- **RQ2 (Controllability):** Does WorkLoadWeaver respond predictably and interpretably to user-controlled knobs?
- **RQ3 (Description Faithfulness):** Does WorkLoadWeaver faithfully reflect a natural language domain description in the generated workload?
- **RQ4 (Human Checkpoints):** Does human checkpoints improve the integration of domain knowledge that is not inferable from schema and templates alone?

Setup. All experiments use TPC-H (scale factor 10) as the benchmark template set with all 22 query templates. The three LLM agents (Persona, Parameter, Event) are each powered by Gemini 2.5 Pro via litellm, though any LLM can be swapped in through the configuration file. For RQ1 and RQ2 we generate traces of approximately 1,000 queries each over a simulated 24-hour window. We compare against two baselines: **B1**, which draws templates uniformly at random from all 22 TPC-H templates with parameters sampled uniformly from their domains; and **B2**, a CAB-style [20] frequency-weighted sampler with Poisson inter-arrival times but no session or persona modeling.

Agent Outputs. We briefly summarize the workload specification produced by the three LLM agents, as these decisions ground all subsequent experiments.

Persona Agent. The Persona Agent proposed four distinct personas for TPC-H, each with a natural-language rationale, a subset of relevant templates, a temporal activity profile, and a session configuration. The *Strategic Analyst* focuses on long-term trends and profitability (8 templates, ∼8 queries/session, business hours weekly), chaining pricing reports (Q1) with supply cost analyses (Q2) and market share queries (Q8). The *Operations Manager* monitors day-to-day supply chain logistics (9 templates including Q2, Q3, Q4, Q11, Q12, Q15, Q16, Q20, Q21; ∼5 queries/session, business hours). The *CRM Analyst* targets customer segmentation and retention (5 templates: Q3, Q10, Q13, Q18, Q22; ∼6 queries/session, business hours mid-week), while the *Automated Dashboard* runs a fixed set of periodic summary queries (Q1, Q4, Q5, Q14) at regular 4-hour intervals with single-query sessions. Each persona also received a persona-conditioned sparse transition matrix over templates; the Strategic Analyst's matrix, for example, assigns near-deterministic transitions from Q1→Q7, Q5→Q8, and Q6→Q6 (self-loop for parameter refinement), with remaining transitions filled with uniform ε-probability.

Parameter Agent. The Parameter Agent produced 59 slot distributions covering all parameters across all 22 templates. Of these, 34 are categorical distributions (e.g., region names, ship modes, part types) and 25 are continuous distributions (21 uniform, 3 categorical over date values, 1 normal for Q6 quantity). The agent also produced 6 dependency DAGs capturing inter-parameter constraints: Q7 and Q12 enforce distinctness between their two entity parameters (nation1≠nation2, shipmode1≠shipmode2), Q8 enforces that the selected nation belongs to the selected region, Q16 enforces distinct sizes across its 8 size parameters, Q19 enforces distinct brands, and Q22 enforces 7 distinct country codes.

Event Agent. The Event Agent introduced 6 events across all three categories: (i) *Arrival events*: an *end-of-quarter rush* (3× analyst activity for 10 days) and a *regional economic boom* (1.5× regional orders for 30 days); (ii) *Mutation events*: a *new product launch* (inserts new parts and expands brand parameters) and *daily stock replenishment* (updates inventory quantities); (iii) *Shock events*: a *flash sale* (20× promotional orders, 5× all persona activity for 24 hours) and a *supply chain disruption* (zeroes out a supplier's inventory, delays open shipments). Each event includes triggered queries dispatched to specific personas—e.g., the supply chain disruption triggers Q21 (suppliers who kept orders waiting), Q2 (minimum

Table 2: RQ1: Session structure comparison. Transition entropy is the conditional Shannon entropy of the empirical template transition matrix (nats); higher values indicate more uniform (less structured) transitions. Session length is the mean number of queries per session.

Metric	B1	B2	WorkLoadWeaver
Transition entropy (nats)	2.84	2.83	**1.10**
Mean session length	1.0	1.0	**4.96**
Session length std	0.0	0.0	8.30

Table 3: RQ2a: Persona weight sweep. KL divergence of the template frequency distribution from the pure-Analyst distribution decreases monotonically as the Analyst fraction increases.

Analyst fraction	0.1	0.3	0.5	0.7	0.9
D_{KL}	0.097	0.071	0.026	0.014	0.008

cost supplier), and Q11 (important stock identification) for the Operations Manager.

RQ1: Session Structure. Table 2 compares B1, B2, and WorkLoadWeaver on two structural metrics: template transition entropy and mean session length.

Both baselines exhibit transition entropy close to the theoretical maximum of $\log(22) \approx 3.09$ nats, confirming that their template sequences are effectively IID uniform. In contrast, WorkLoadWeaver achieves a transition entropy of 1.10 nats—a 2.6× reduction—reflecting the structured template transition patterns induced by persona-conditioned transition matrices. At the session level, B1 and B2 generate independent queries (session length = 1 by construction), while WorkLoadWeaver produces realistic multi-query sessions with a mean length of 4.96 queries (std = 8.30), capturing coherent analytical workflows. For example, the Strategic Analyst persona chains pricing summary reports (Q1) with supply cost analyses (Q2) and market share queries (Q8), while the Automated Dashboard persona cycles through periodic reporting templates.

RQ2: Controllability. Table 3 and Table 4 show results for two knob-sweep experiments.

RQ2a (Persona Weight Sweep). We vary the Strategic Analyst persona weight from 10% to 90% (redistributing the remaining weight uniformly among other personas) and measure KL divergence of the resulting template frequency distribution from the pure-Analyst distribution. The KL divergence decreases smoothly and monotonically from 0.097 at 10% to 0.008 at 90%, confirming that WorkLoadWeaver responds predictably to persona weight adjustments. As the Analyst fraction increases, the template frequency distribution converges toward the pure-Analyst reference distribution, demonstrating fine-grained controllability over the workload's composition.

RQ2b (Event Intensity Sweep). We fix the persona mix and scale event occurrence rates from 0× (no events) to 4×, measuring the template frequency shift (KL divergence from the no-event baseline). The template shift increases monotonically from 0.000 to 0.014

Table 4: RQ2b: Event intensity sweep. Template shift (KL divergence from the no-event baseline) increases monotonically with event intensity.

Multiplier	0×	1×	2×	4×
Template shift (D_{KL})	0.000	0.007	0.011	0.014

Table 5: RQ3: Description faithfulness. No-description condition is identical across scenarios.

Scenario	Condition	Concurrency	Temporal CV
Startup	w/ description	13.66	0.180
Retail Analyt.	w/ description	6.82	0.404
Data Pipeline	w/ description	20.24	0.137
w/o description (all scenarios)		8.26	0.353

as event intensity rises, confirming that event-triggered queries measurably alter the workload's template composition. Together, RQ2a and RQ2b demonstrate that WorkLoadWeaver exposes independent, predictable control over two orthogonal dimensions of workload structure: persona-driven template selection and event-driven query bursts.

RQ3: Description Faithfulness. To evaluate whether the domain description input meaningfully steers workload generation, we author three domain descriptions of contrasting character: (i) a startup with 20 users and irregular schedules, (ii) a retail analytics team operating on business-hours cadence with heavy aggregation queries, and (iii) a data engineering pipeline with low concurrency and bursty batch arrivals. For each description, we run WorkLoadWeaver with and without it (schema and dataset held fixed). We measure two proxy metrics: *concurrency alignment* (mean active sessions vs. the stated user count) and *temporal regularity* (coefficient of variation of $\lambda_p(t)$, where low CV corresponds to activity spread uniformly across the day and high CV to traffic concentrated in a narrow window). We report both conditions—with and without description—for each metric across all three scenarios. We expect description-conditioned output to move each metric in the direction indicated by the description, while the no-description baseline remains at a neutral default.

Table 5 reports both metrics under both conditions for each scenario. The no-description baseline yields consistent outputs across all three scenarios (concurrency: 8.26, CV: 0.353), serving as a neutral reference. In both the startup and retail scenarios, description-conditioned outputs move each metric in the expected direction. For the *startup* scenario, concurrency rises 65% toward the stated 20-user count and CV drops to 0.180, consistent with a timezone-diverse team spread across the day. For the *retail analytics* scenario, concurrency falls (smaller focused team) and CV rises to 0.404, consistent with traffic concentrated in the stated business-hours window. For the *data pipeline* scenario, the Persona Agent correctly captures the machine-driven character of the description by generating scheduled batch personas with smooth, uniform arrival profiles (CV of 0.137, the lowest of all conditions); the higher apparent concurrency reflects the heavy per-job query volume of

Table 6: RQ4: Recovery of hidden domain facts. Each checkpoint contributes 2 facts.

Condition	Recov. Score	CP1 (/2)	CP2 (/2)	CP3 (/2)
Auto	0.000	0	0	0
Upfront	0.167	1	0	0
Summary-guided	0.833	1	2	2
Oracle	1.000	2	2	2

Table 7: RQ4: Structural comparison between Summary-guided and Oracle.

Metric	Summary-guided	Oracle
Trans. entropy (nats)	0.976	0.260
Mean session length	16.55 ± 10.56	15.53 ± 12.38

batch runs rather than many simultaneous human users. Across the two primary scenarios, all four metric-direction predictions are confirmed, demonstrating that the domain description provides meaningful signal beyond the schema and dataset alone.

RQ4: Checkpoint-Guided Domain Knowledge Integration. To evaluate whether human checkpoints make external domain knowledge more actionable, we define six hidden facts that are compatible with TPC-H but not inferable from schema and templates alone, assigning two facts to each checkpoint. The six facts are: (CP1) the Strategic Analyst works exclusively during US East Coast business hours; the Operations Manager runs 24/7 across time zones. (CP2) `order_date` parameters follow a recency-weighted distribution with 80% of queries targeting the last 90 days; `discount` values cluster near promotional thresholds (0.05, 0.10). (CP3) a nightly batch load at 2am triggers a 30-minute burst of validation queries; a quarterly earnings freeze halts all write events for 48 hours. We compare four conditions: *Auto* (no facts provided), *Upfront* (all six facts given once to Agent 1), *Summary-guided* (facts disclosed only at the relevant checkpoint, with one regeneration), and *Oracle* (facts written directly into intermediate sub-specs). We report Recovery Score, the fraction of hidden facts reflected in the final specification, as the primary metric. Table 6 reports the recovery score with recovery counts out of 2 (2 facts per checkpoint).

Auto recovers none of the six facts, which confirms that the chosen facts are genuinely underdetermined by the benchmark inputs. *Upfront* improves to a Recovery Score of 0.167, but its single recovered fact is at CP1 (1/2); it recovers 0/2 at both CP2 and CP3, suggesting that facts front-loaded into the initial prompt are diluted before later agents act on them. *Summary-guided* reaches 0.833, recovering 1/2 at CP1 and 2/2 at both CP2 and CP3, because each fact is disclosed at the checkpoint where the responsible agent can directly incorporate it. The gap between *Upfront* and *Summary-guided*, despite both receiving identical information, shows that *when* feedback is delivered matters as much as *whether* it is delivered, validating our staged human-in-the-loop design.

Table 7 compares structural metrics between Summary-guided and Oracle. Summary-guided produces session lengths comparable to Oracle (16.55 vs. 15.53 queries/session), confirming that the checkpoint mechanism preserves coherent session structure. Transition entropy is higher under Summary-guided (0.976 vs. 0.260 nats), reflecting the inherent imprecision of LLM-mediated regeneration relative to direct sub-spec editing; both conditions nonetheless remain substantially more structured than the IID baselines (2.84 nats).

6 Discussion and Future Work

WorkLoadWeaver currently evaluates on TPC-H only and measures structural and controllability properties rather than fidelity against production traces, which are unavailable by assumption. We outline two directions for future work.

Downstream utility via workload-sensitive optimization. A natural next step is to evaluate whether WorkLoadWeaver-derived workloads lead to meaningfully different decisions in downstream optimization tasks such as index selection and cost model training. Because persona-conditioned parameter distributions concentrate queries on narrow date ranges and specific template subsets, we expect optimization decisions derived from WorkLoadWeaver traces to differ structurally from those derived from uniform IID baselines, with an asymmetric cost penalty when the wrong workload is used. More broadly, because WorkLoadWeaver's intensity profiles are Fourier-parameterized, perturbation rules can generate workload pairs at calibrated shift severities, enabling systematic evaluation of whether self-driving learners (e.g. learned query optimizers [13], adaptive index selectors, and cost model retrainers) detect and adapt to workload shifts or degrade silently .

Data-coupled workload generation. WorkLoadWeaver assumes that queriers are separate from the data they query. In domains such as smart spaces and ride hailing [6], the querier *is* a data point: a user waiting for a taxi generates queries whose frequency depends on local wait times and vehicle supply, and may expand searches to neighboring regions as wait times grow; meanwhile, more users in a region means more queries from that region. Query generation and data generation must therefore proceed hand in hand. Extending WorkLoadWeaver to co-evolve data and queries by applying the specification model to benchmarks such as SmartBench [6] is a promising direction that would broaden its applicability beyond decision-support workloads.

7 Conclusion

We presented WorkLoadWeaver, a benchmark-grounded pipeline that constructs workload specifications from query template benchmarks through three sequential LLM agents, each producing a provenance-tagged Assumption Summary for human review. Experiments on TPC-H show that WorkLoadWeaver produces traces with 2.6× lower transition entropy than IID baselines, responds predictably to persona and event knobs, faithfully reflects natural-language domain descriptions, and recovers hidden domain facts significantly more effectively when feedback is staged at the relevant checkpoint than when front-loaded at the first agent.

Acknowledgments

This work was supported in part by NSF awards 2540011, 2420846, 2419844, 2245372, 2113930, 2133391, 2008993, and 1952247, and by HPI@UCI. Sarvesh acknowledges the financial support of the United States–India Education Foundation and the Fulbright program.

References

[1] Christoph Brücke, Philipp Härtling, Rodrigo D Escobar Palacios, Hamesh Patel, and Tilmann Rabl. 2023. TPCx-AI - An Industry Standard Benchmark for Artificial Intelligence and Machine Learning Systems. *Proc. VLDB Endow.* 16, 12 (Aug. 2023), 3649–3661. doi:10.14778/3611540.3611554

[2] Yegor Denisov-Blanch, Joshua Kazdan, Jessica Chudnovsky, Rylan Schaeffer, Sheng Guan, Soji Adeshina, and Sanmi Koyejo. 2026. Consensus is Not Verification: Why Crowd Wisdom Strategies Fail for LLM Truthfulness. arXiv:2603.06612 [cs.CL] https://arxiv.org/abs/2603.06612

[3] Bailu Ding, Surajit Chaudhuri, Johannes Gehrke, and Vivek R. Narasayya. 2021. DSB: A Decision Support Benchmark for Workload-Driven and Traditional Database Systems. *Proc. VLDB Endow.* 14, 13 (2021), 3376–3388.

[4] Dhrubajyoti Ghosh, Peeyush Gupta, Sharad Mehrotra, and Shantanu Sharma. 2022. A Case for Enrichment in Data Management Systems. *SIGMOD Rec.* 51, 2 (July 2022), 38–43. doi:10.1145/3552490.3552497

[5] Dhrubajyoti Ghosh, Peeyush Gupta, Sharad Mehrotra, Roberto Yus, and Yasser Altowim. 2022. JENNER: Just-in-time Enrichment in Query Processing. *Proc. VLDB Endow.* 15, 11 (2022), 2666–2678. doi:10.14778/3551793.3551822

[6] Peeyush Gupta, Michael J. Carey, Sharad Mehrotra, and oberto Yus. 2020. SmartBench: a benchmark for data management in smart spaces. *Proc. VLDB Endow.* 13, 12 (July 2020), 1807–1820. doi:10.14778/3407790.3407791

[7] Konstantinos Karanasos, Matteo Interlandi, Doris Xin, Fotis Psallidas, Rathijit Sen, Kwanghyun Park, Ivan Popivanov, Supun Nakandal, Subru Krishnan, Markus Weimer, Yuan Yu, Raghu Ramakrishnan, and Carlo Curino. 2019. Extending Relational Query Processing with ML Inference. arXiv:1911.00231 [cs.DB] https://arxiv.org/abs/1911.00231

[8] Song-Hee Kim and Ward Whitt. 2014. Are Call Center and Hospital Arrivals Well Modeled by Nonhomogeneous Poisson Processes? *Manufacturing & Service Operations Management* 16, 3 (2014), 464–480.

[9] Yubin Kim, Ken Gu, Chanwoo Park, Chunjong Park, Samuel Schmidgall, A. Ali Heydari, Yao Yan, Zhihan Zhang, Yuchen Zhuang, Mark Malhotra, Paul Pu Liang, Hae Won Park, Yuzhe Yang, Xuhai Xu, Yilun Du, Shwetak Patel, Tim Althoff, Daniel McDuff, and Xin Liu. 2025. Towards a Science of Scaling Agent Systems. arXiv:2512.08296 [cs.AI] https://arxiv.org/abs/2512.08296

[10] Peter Kraft, Daniel Kang, Deepak Narayanan, Shoumik Palkar, Peter Bailis, and Matei Zaharia. 2020. Willump: A Statistically-Aware End-to-end Optimizer for Machine Learning Inference. arXiv:1906.01974 [cs.DB] https://arxiv.org/abs/1906.01974

[11] Jiale Lao and Immanuel Trummer. 2026. SQLBarber: A System Leveraging Large Language Models to Generate Customized and Realistic SQL Workloads. *Proc. ACM Manag. Data* 4, 1 (2026). https://arxiv.org/abs/2507.06192

[12] Lin Ma, Dana Van Aken, Ahmed Hefny, Gustavo Mezerhane, Andrew Pavlo, and Geoffrey J. Gordon. 2018. Query-based Workload Forecasting for Self-Driving Database Management Systems. In *Proc. SIGMOD*. 631–645.

[13] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Nesime Tatbul, Mohammad Alizadeh, and Tim Kraska. 2021. Bao: Making Learned Query Optimization Practical. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD)*. 1275–1288. doi:10.1145/3448016.3452838

[14] Beng Chin Ooi et al. 2025. NeurDB: On the Design and Implementation of an AI-Powered Autonomous Database. In *15th Biennial Conference on Innovative Data Systems Research (CIDR)*. https://vldb.org/cidrdb/papers/2025/p29-zhao.pdf

[15] Andrew Pavlo, Gustavo Angulo, Joy Arulraj, Haibin Lin, Jiexi Lin, Lin Ma, Prashanth Menon, Todd C. Mowry, Matthew Perron, Ian Quah, Siddharth Santurkar, Anthony Tomasic, Skye Toor, Dana Van Aken, Ziqi Wang, Yingjun Wu, Ran Xian, and Tieying Zhang. 2017. Self-Driving Database Management Systems. In *8th Biennial Conference on Innovative Data Systems Research (CIDR)*. http://cidrdb.org/cidr2017/papers/p42-pavlo-cidr17.pdf

[16] Andy Pavlo, Matthew Butrovich, Lin Ma, Prashanth Menon, Wan Shen Lim, Dana Van Aken, and William Zhang. 2021. Make Your Database System Dream of Electric Sheep: Towards Self-Driving Operation. *Proc. VLDB Endow.* 14, 12 (2021), 3211–3221. doi:10.14778/3476311.3476411

[17] Tobias Schmidt, Viktor Leis, Peter Boncz, and Thomas Neumann. 2025. SQLStorm: Taking Database Benchmarking into the LLM Era. *Proc. VLDB Endow.* 18, 11 (2025), 4144–4157. https://www.vldb.org/pvldb/vol18/p4144-schmidt.pdf

[18] Transaction Processing Performance Council (TPC). [n. d.]. TPC Benchmark™H (TPC-H). https://www.tpc.org/tpch/. Accessed 2026-03-10.

[19] Transaction Processing Performance Council (TPC). 2021. TPC Benchmark™DS (TPC-DS) Standard Specification, Version 3.2.0. https://www.tpc.org/tpc_documents_current_versions/pdf/tpc-ds_v3.2.0.pdf. Accessed 2026-03-10.

[20] Alexander van Renen and Viktor Leis. 2023. Cloud Analytics Benchmark. *Proc. VLDB Endow.* 16, 6 (2023), 1413–1425. https://www.vldb.org/pvldb/vol16/p1413-renen.pdf

[21] Chengcheng Wan, Yiwen Zhu, Joyce Cahoon, Wenjing Wang, Katherine Lin, Sean Liu, Raymond Truong, Neetu Singh, Alexandra M. Ciortea, Konstantinos Karanasos, and Subru Krishnan. 2023. Stitcher: Learned Workload Synthesis from Historical Performance Footprints. In *Proc. EDBT*. 417–423. https://openproceedings.org/2023/conf/edbt/paper-19.pdf

[22] Johannes Wehrstein, Roman Heinrich, Mihail Stoian, Skander Krid, Martin Stemmer, Andreas Kipf, Carsten Binnig, and Muhammad El-Hindi. 2025. Redbench: Workload Synthesis From Cloud Traces. arXiv:2511.13059. https://arxiv.org/abs/2511.13059

[23] Yan Zhou et al. 2025. PBench: Workload Synthesizer with Real Statistics for Cloud Analytics Benchmarking. *Proc. VLDB Endow.* 18, 11 (2025), 3883–3895. doi:10.14778/3749646.3749661