Querying Everything Everywhere All at Once

Supervaluationism for the Agentic Lakehouse

Jacopo Tagliabue
jacopo.tagliabue@bauplanlabs.com
Bauplan Labs
New York, USA

Aldrin Montana
aldrin.montana@bauplanlabs.com
Bauplan Labs
New York, USA

Abstract

Agentic analytics is turning the lakehouse into a multi-version system: agents materialize data and insights on alternate branches, while users may need answers before those branches are merged or discarded. We present a system that answers questions *across* branches rather than *at* a single snapshot. The system evaluates queries under supervaluationary semantics and presents agreement and disagreement in a form that supports human review. We compare two implementations of a multi-branch query engine–*ad hoc* and *native*–to show that shared-subplan reuse and short-circuit evaluation make the abstraction practical. We release the code as an open reference implementation for multi-branch querying in OLAP systems.

CCS Concepts

• **Information systems** → **Data management systems**; • **Computing methodologies** → **Intelligent agents**; • **Theory of computation** → **Logic**.

Keywords

agentic analytics, lakehouse, data branching, supervaluationism, text-to-SQL, DataFusion

ACM Reference Format:

Jacopo Tagliabue and Aldrin Montana. 2026. Querying Everything Everywhere All at Once: Supervaluationism for the Agentic Lakehouse. In *Proceedings of Supporting Our AI Overlords Workshop (CAIS '26).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

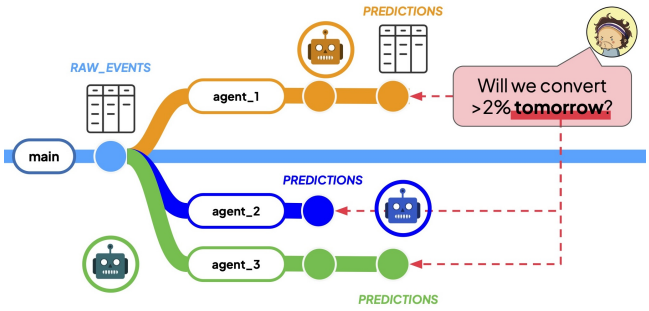

Figure 1: *ACME Inc.* scenario: three agentic pipelines run concurrently on data branches, producing three incompatible *prediction* tables. Can we answer questions without assuming a canonical data representation?

1 Introduction

> "Falsehood is never in words; it is in things."
> I. Calvino, *Invisible Cities*

In the year of the (over)lord(s) [9] *one*, a data engineer at *ACME Inc.* kicks off a purchase prediction job, resulting in three agents swarming to answer the question, "Will we convert 2% tomorrow?" This scenario, illustrated in Fig. 1, depicts an online shop using agents for reporting and forecasting. While a *review-then-merge* flow would be the natural conclusion (Fig. 2), verification is becoming the bottleneck: if a non-technical user wants to know *now* about tomorrow's sales, unaware of the data lifecycle, how can the reports be verified for correctness (and thus reliable for making business decisions)? Analyses from multiple agents often disagree, but only in part. For example, two agents predicting tomorrow's sales may disagree on the exact figure, yet they may both agree that revenue grew for a particular customer segment. Can a system meaningfully answer this question without *assuming* that technical review has already happened?

We argue that a non-classical logic, *supervaluationism*, applies naturally to this setting: rather than waiting for, or forcing, a single canonical version, we can reason *across* versions to draw conclusions [17]. In particular, we describe a system on top of a cloud lakehouse [18] that aggregates across data versions to answer user questions, instead of using the per-version querying model of standard OLAP engines. In contrast to time travel analyses that answer a "query at snapshot t", we use supervaluationism to answer a "query across all data branches". We summarize our contributions as follows:

(1) we ground our use case using an agentic lakehouse at scale [13]. Far from being a theoretical possibility, data agents *today* are already generating too many data branches for humans to review one by one: hundreds of thousands of *data branches* [16] are created in our agentic lakehouse in production every week.
(2) we provide intuitive and formal definitions for supervaluationary semantics and showcase the difference between an *ad hoc* and *native* implementation of the relevant query plans;

CAIS '26, San Jose, CA

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

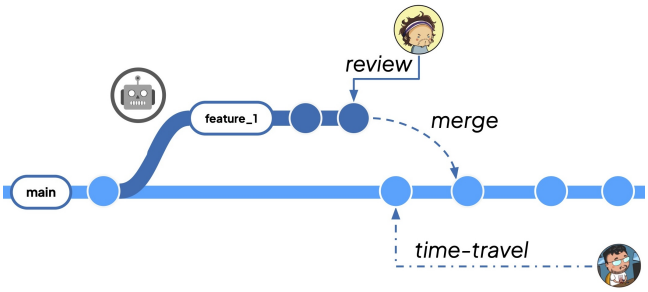

Figure 2: Human-in-the-loop control over data branches: review before merge, recover through time travel.

(3) we discuss how answers should be presented to facilitate human reasoning under uncertainty, and highlight limitations and possible extensions based on user feedback.

We include *preliminary* experiments, but our goal is to share with the community a technical innovation motivated by user-facing interactivity concerns. We believe this work to be interesting to a wide set of practitioners, as it addresses a full vertical slice: from business user needs and organizational constraints down to system-level optimizations inside a query engine. Furthermore, at the moment of writing, no major *OLAP* platform has branching APIs or support for arbitrary transactions, making direct comparisons impossible[1]: by releasing code[2] grounded in industry open standards, including *D3.js*, *Iceberg* [2], and *DataFusion* [7], we hope to foster community interest in agent-first architectures. Finally, mostly as a nod to our younger selves, we are delighted to observe that conflicting data were the motivating examples for Belnap's seminal work [3] in multi-valued logic.[3]

2 Motivation

The rise of AI agents and patterns such as *Ralph* [6] highlight a novel challenge for OLAP systems: instead of a slow-moving code base managed by a few data engineers working in sequence, we now have massively parallel experimentation done by swarms of untrusted agents [9]. As experimental code lives in code branches, **data producers** should materialize experimental data assets in *data branches*, i.e., copy-on-write versions of the lakehouse with strong isolation guarantees [14].

While the human-in-the-loop *merge* is the standard process to publish insights to the "production data branch" (Fig. 2), traces from Bauplan's production deployment – with over a million data branches created in the last quarter – show that multiple versions of the same table are likely to coexist at any given moment in time. There are three recurring causes: (1) data pipelines may be long-running processes, (2) data pipelines must exist until they are reviewed (and subsequently merged), and (3) disagreement (conflicts) are not always accidental.

First, for long-running data pipelines, branches provide a place to materialize partial or intermediate outputs without leaking them into the production data branch. The longer lived a data pipeline is, or the more alternate branches there are to compute, the more likely it is that a business user will want to ask a question about the data before it has been merged. Second, there is a natural time gap between when a data pipeline completes and when it is reviewed and merged. This situation is then amplified as agents generate candidate changes much faster than humans can inspect, compare, and merge them, leading to more data pipelines that are complete and awaiting review. Third, some analytical requests are genuinely polysemic: for example, "best customers tomorrow" may mean highest expected revenue, highest conversion probability, highest churn risk, or some stakeholder-specific combination of these notions.

This is not the only acceleration happening. On the **data consumer** side, the slow "business to analyst to engineering" handover is being quickly dismantled by text-to-SQL and data agents [5]. The user experience quickly degrades, however, as questions are asked on tables existing in multiple active branches: as the business user has neither the knowledge nor the access to move the underlying data lifecycle forward, she is once again slowed down by processes and infrastructure she does not understand. In other words, our design challenge is not only to query across branches efficiently, but to present uncertainty in a form that supports review. If the system collapses all disagreement into a single scalar too early, it removes precisely the information a human needs. If it surfaces only raw per-branch outputs, it overwhelms the user and recreates the original bottleneck.

Our design shows that another way is possible: the coupling of data producers and data consumers can be loosened if we can reason consistently across multiple versions of the same tables. In other words, if we can query *everything everywhere all at once.*

3 System overview

Our setup starts from a real-world production lakehouse with Git-like primitives over Iceberg tables: *commits*, *branches*, and *merges* [16]. We discuss the role of UI for human review below (Section 5), and focus now on the back-end modules on the *query path*, i.e., going from English questions to answers while reasoning across branches.

3.1 Architecture

The overall architecture is depicted in Fig. 3. At the bottom, a branching lakehouse (*Bauplan* [15]) allows data agents to write data pipelines concurrently (Fig. 1); at the top, a *D3-based application* provides a GUI, backed by a *FastAPI* server. The novel part is the reasoning engine in the middle: an LLM-powered text-to-SQL translation layer and a modified DataFusion OLAP engine, implementing the supervaluationary semantics (Section 3.2) when querying data branches.

For the text-to-SQL sub-module, we implement a narrow but complete interface for interactive analytics over a known schema. The module follows best practices from recent text-to-SQL work [5]: schema augmentation, constrained prompt templates, feedback-driven regeneration, and plan-first validation. Instead of running arbitrary SQL directly from an LLM completion, the system attempts to produce a valid query plan first and fails early when the

[1] For example, the three largest lakehouses: *Databricks*, *Snowflake*, and *Fabric*. A few OLTP solutions exist, but they cover different use cases than data pipelines and data science.

[2] https://github.com/BauplanLabs/AI-for-Distributed-System-Design

[3] "The computer is to be envisioned as obtaining the data (...) from a variety of sources, all of which indeed may be supposed to be on the whole trustworthy, but none of which can be assumed to be that paragon of paragons, a universal truth-teller".

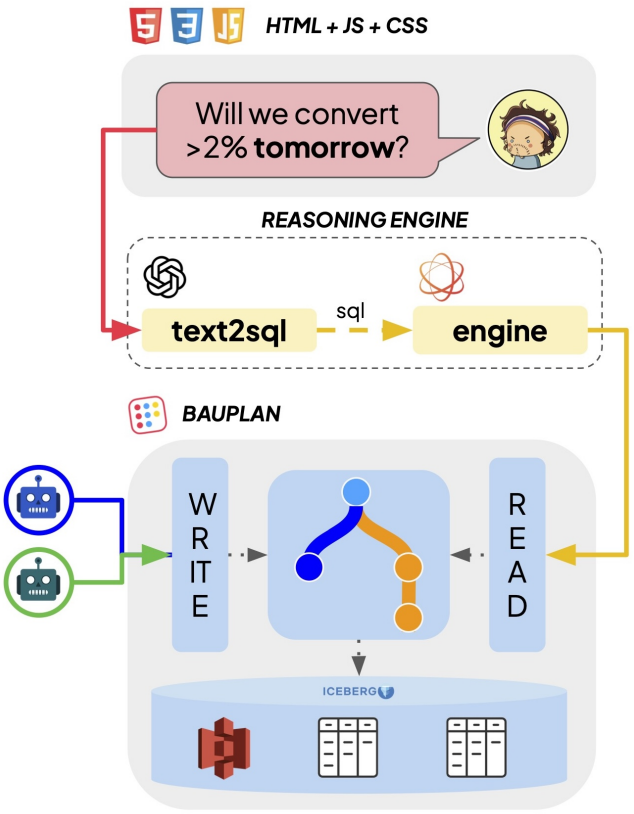

Figure 3: Architecture of the system. Agentic pipelines write predictions into isolated *data branches* in Bauplan. The UI translates English questions into SQL, then executes them using a modified OLAP engine that evaluates answers across branches under supervaluationary, glut-tolerant semantics.

generated query is ill-typed or unsupported. Since multi-branch execution may magnify bad queries, the system *must* validate the query before dispatching it.

3.2 Supervaluationism

We first explain how non-classical semantics apply to reasoning across branches. We then describe the implementation inside a state-of-the-art engine and state known limitations and directions for future work.

3.2.1 Logic. In our scenario (Fig. 1), *ACME Inc.* is an e-commerce company where data agents are tasked to compute possibly competing predictions on future shoppers' behavior. Consider the following statements:

(1) Conversion is >2%.

(2) It is not the case that conversion is >2%.

(3) Conversion is >2% and it is not the case that conversion is >2%.

(4) Conversion is >2% OR revenues are non-negative.

1 is true *in some branch*, 2 *in some other branch*, but 3 is never true: *in some branch* does not pass through the truth-functional conjunction [8]. Notably, not anything goes, as we still have enough logical structure to assert 4, which is true even if half of it is only true *in some branch*. In a *supervaluationary*[4] approach, 1 is a truth *glut* [17], a statement to which the model[5] assigns *multiple* truth values, which is classically impossible. At evaluation time, if we get the same outcome no matter how we "clear the gluts", then the gluts do not matter and sentences may be true or false *simpliciter*. For example, assuming revenues are non-negative, 4 is true in any "contraction" of the model, as it does not matter how a particular branch clears the glut of the first disjunct. Armed with this core intuition, we now tackle the engineering challenges; we refer readers to Appendix A for a formal sketch.

3.2.2 Engineering. As supervaluationary semantic assignments are parasitic over per-branch assignments, a first implementation strategy suggests itself: evaluate the query Q independently on each branch and concatenate the per-branch outputs, tagging each row with a branch identifier column. For instance, "How many buyers do we expect tomorrow?" becomes a `SELECT COUNT(*)` executed once per branch, producing one scalar result per branch. This implementation is bolted on top of an existing engine implementation and referred to as the *ad hoc* engine.

Table 1 contains representative business questions for our *ACME Inc.* scenario. We divide them by result type — number, boolean, or list — which determines the outer semantic logic: numbers are shown per branch with a compact summary; booleans yield a supervaluationary verdict, namely true, false, or mixed; lists are compared via Git-style *diffs*.

There are reasons to be dissatisfied with this ad hoc approach. First, answering questions such as $Q3$ involves a `JOIN`, and the ad hoc strategy repeats the same work B times even when some inputs are identical across branches. More subtly, the supervaluationary approach uncovers optimizations that depend on the result type. We therefore implement a *native* multi-branch engine that dispatches to specialized execution paths.

For aggregate and join queries, the native engine uses a custom DataFusion `TableProvider` that exposes all branch variants of a table as a single virtual relation with an injected `__branch_id` column. The entire query plan runs in a single Rust process, giving the optimizer full visibility for predicate pushdown, projection pruning, and shared-subplan reuse. For boolean queries such as $Q4$, the native engine also supports a short-circuit evaluator: it runs per-branch evaluations in parallel and terminates as soon as two branches disagree, since the verdict, mixed, is then determined without scanning the remaining branches. Such early termination is difficult to express as static SQL rewriting, but natural to implement in the multi-branch coordinator. We measure the performance gap between the ad hoc and native strategies in Section 4.

4 Experiments

To support the system design, we report a small set of targeted experiments on representative query types. Our goal is not exhaustive benchmarking, but to show that native multi-branch execution

[4]Since we are dealing with truth gluts and not gaps, "subvaluationary" is technically more appropriate: with a slight abuse of notation, we choose to use the more familiar term throughout the paper.

[5]"Model" in the sense of model theory, not AI: we rely on the context to disambiguate.

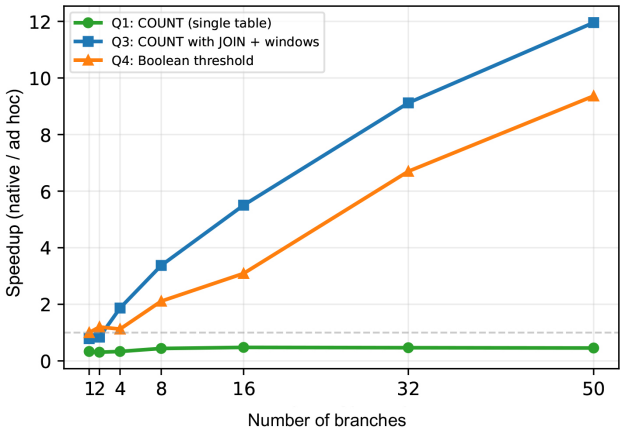

Figure 4: Speedup of the native engine over the ad hoc strategy as a function of branch count. The JOIN query with window functions ($Q3$) benefits from shared-table reuse; the boolean query ($Q4$) achieves near-constant time via short-circuit evaluation.

already yields measurable benefits for typical agentic access patterns [9]. We pick representative queries, and run them over test tables varying the number of branches $B \in \{1, \ldots, 50\}$ (Appendix B). Fig. 4 shows the speedup. For simple single-table counts ($Q1$), the unified plan's coordination overhead outweighs the benefit of a single plan, with the ad hoc engine beating the native one. For $Q3$ — a `JOIN` with window functions over the shared dimension table — the picture reverses: the ad hoc engine recomputes the expensive CTE per branch, while the native engine computes it once and reuses the hash join, yielding 12.0× at B=50. For boolean queries, the supervaluationary semantics enables a short-circuit optimization: once two branches disagree, the verdict is determined, so $Q4$ stays flat at ∼11 ms regardless of B.

5 Human-computer interaction

The point of reasoning across branches is not merely to avoid picking a single version too early, but to give a human enough structure to decide what to do next. The system therefore chooses the presentation by result type. For numeric questions, agreement yields a single scalar; disagreement yields `UNCLEAR` plus a compact min/-max/mean summary and per-branch values. For boolean questions, agreement yields `YES` or `NO`; disagreement yields `UNCLEAR` plus explicit counts of supporting and refuting branches. For set-valued questions, the system separates consensus items from branch-specific disagreement and offers a Git-style diff.

Table 1 summarizes the review surface exposed by the prototype. The goal is to preserve enough provenance for a human to decide whether a branch is ready to merge, should remain under review, or should be discarded, without forcing the user to inspect raw outputs from every branch.

Table 1: Representative questions by result type

ID	User-facing question	Result type
Q1	How many customers will buy tomorrow?	Number
Q2	What is the expected revenue tomorrow?	Number
Q3	How many smartphone shoppers will buy tomorrow?	Number
Q4	Is tomorrow's conversion rate above τ?	Boolean
Q5	Will *John Doe* buy tomorrow?	Boolean
Q6	Who will buy tomorrow?	List
Q7	Who is undecided?	List

6 Related work

Database branching. Git-like semantics for data have been explored before, mostly for human workflows: for example, Dolt [4] supports SQL in an OLTP setup. Industry lakehouses built on Iceberg per-table evolution support `QUERY AT t`, but lack the API expressivity to even *formulate* our research question. Building on a system supporting transactions across tables, languages, and operations [16] unlocks a richer UX, backed by a genuinely novel query semantics.

Non-classical logic. The basic insights for reasoning over "precisifications" come from supervaluationism [10], which is easily adapted to gluts in the case of paraconsistency [17]. Our contribution is in mapping the theoretical semantics onto an agentic lakehouse at industry scale, yielding a new end-to-end experience for agentic analytics.

7 Conclusions and future work

The system defines supervaluationary semantics over three target question types, and provides a small set of initial benchmarks (Section 4) to start mapping out the design space. The approach can be generalized in several ways. **At the engine level**, we could push evaluation for more complex, non-scalar queries. **At the user level**, a similar approach could work for on-the-fly analytical jobs in response to polysemic requests; e.g., when reporting on "customer lifetime value", we could reason over multiple precisifications [10] of that concept. **At the formal level**, a complete treatment would model the inter-branch relationship: for example, commit histories could form a modal structure with truth evaluated locally at a world [11]. Broadly speaking, our north star is to treat "quarantined inconsistencies" [8] and "conceptual indeterminacy" [1] as first-class citizens of our interactions with data systems, instead of something we need to "fix" *before* analysis can begin.

Aside from technical contributions, the system effectively defines a novel API for agent-first OLAP systems. Instead of querying tables `AT A SNAPSHOT`, we are moving up in abstraction by querying *the entire lakehouse all at once.* As deployments of data-agent swarms rise [9, 12] and English becomes a primary interface to data, we may benefit from systems that can reason in a principled way over truth gaps and truth gluts alike.

References

[1] Lucía Gómez Álvarez and Brandon Bennett. 2018. Dealing with Conceptual Indeterminacy: A Framework Based on Supervaluation Semantics. In *MedRACER+WOMoCoE@KR*. https://api.semanticscholar.org/CorpusID:53227110

[2] Apache. 2024. Iceberg. https://github.com/apache/iceberg.

[3] Nuel D. Belnap. 1977. A Useful Four-Valued Logic. In *Modern Uses of Multiple-Valued Logic*, J. Michael Dunn and George Epstein (Eds.). Springer Netherlands, Dordrecht, 5–37. doi:10.1007/978-94-010-1161-7_2

[4] DoltHub. 2026. Dolt. https://github.com/dolthub/dolt

[5] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. arXiv:2406.08426 [cs.CL] https://arxiv.org/abs/2406.08426

[6] Geoffrey Huntley. 2026. Ralph Wiggum as a "software engineer"'. https://ghuntley.com/ralph/

[7] Andrew Lamb, Yijie Shen, Daniël Heres, Jayjeet Chakraborty, Mehmet Ozan Kabak, Liang-Chi Hsieh, and Chao Sun. 2024. Apache Arrow DataFusion: A Fast, Embeddable, Modular Analytic Query Engine. In *Companion of the 2024 International Conference on Management of Data* (Santiago AA, Chile) *(SIGMOD '24)*. Association for Computing Machinery, New York, NY, USA, 5–17. doi:10.1145/3626246.3653368

[8] David Lewis. 1982. Logic for Equivocators. *Noûs* 16, 3 (1982), 431–441. http://www.jstor.org/stable/2216219

[9] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, Matei Zaharia, Alvin Cheung, Natacha Crooks, Joseph E. Gonzalez, and Aditya G. Parameswaran. 2025. Supporting Our AI Overlords: Redesigning Data Systems to be Agent-First. arXiv:2509.00997 [cs.AI] https://arxiv.org/abs/2509.00997

[10] Manfred Pinkal (Ed.). 1995. *Precisification Semantics*. Springer Netherlands, Dordrecht, 196–254. doi:10.1007/978-94-015-8445-6_7

[11] Graham Priest. 2008. *An Introduction to Non-Classical Logic: From If to Is* (2 ed.). Cambridge University Press.

[12] Natalie Shapira, Chris Wendler, Avery Yen, Gabriele Sarti, Koyena Pal, Olivia Floody, Adam Belfki, Alex Loftus, Aditya Ratan Jannali, Nikhil Prakash, Jasmine Cui, Giordano Rogers, Jannik Brinkmann, Can Rager, Amir Zur, Michael Ripa, Aruna Sankaranarayanan, David Atkinson, Rohit Gandikota, Jaden Fiotto-Kaufman, EunJeong Hwang, Hadas Orgad, P Sam Sahil, Negev Taglicht, Tomer Shabtay, Atai Ambus, Nitay Alon, Shiri Oron, Ayelet Gordon-Tapiero, Yotam Kaplan, Vered Shwartz, Tamar Rott Shaham, Christoph Riedl, Reuth Mirsky, Maarten Sap, David Manheim, Tomer Ullman, and David Bau. 2026. Agents of Chaos. arXiv:2602.20021 [cs.AI] https://arxiv.org/abs/2602.20021

[13] Weiming Sheng, Jinlang Wang, Manuel Barros, Aldrin Montana, Jacopo Tagliabue, and Luca Bigon. 2026. Building a Correct-by-Design Lakehouse. Data Contracts, Versioning, and Transactional Pipelines for Humans and Agents. arXiv:2602.02335 [cs.DC] https://arxiv.org/abs/2602.02335

[14] Jacopo Tagliabue, Federico Bianchi, and Ciro Greco. 2025. Trustworthy AI in the Agentic Lakehouse: from Concurrency to Governance. arXiv:2511.16402 [cs.AI] https://arxiv.org/abs/2511.16402

[15] Jacopo Tagliabue, Tyler Caraza-Harter, and Ciro Greco. 2024. Bauplan: Zero-copy, Scale-up FaaS for Data Pipelines. In *Proceedings of the 10th International Workshop on Serverless Computing* (Hong Kong, Hong Kong) *(WoSC10 '24)*. Association for Computing Machinery, New York, NY, USA, 31–36. doi:10.1145/3702634.3702955

[16] Jacopo Tagliabue and Ciro Greco. 2025. Safe, Untrusted, "Proof-Carrying" AI Agents: toward the agentic lakehouse. arXiv:2510.09567 [cs.AI] https://arxiv.org/abs/2510.09567

[17] Achille C. Varzi. 1997. Inconsistency Without Contradiction. *Notre Dame Journal of Formal Logic* 38, 4 (1997), 621–639. doi:10.1305/ndjfl/1039540773

[18] Matei A. Zaharia, Ali Ghodsi, Reynold Xin, and Michael Armbrust. 2021. Lakehouse: A New Generation of Open Platforms that Unify Data Warehousing and Advanced Analytics. In *Conference on Innovative Data Systems Research*.

A Supervaluationism: formal remarks

We give a model-theoretic sketch tailored to KPI-style sentences. The motivating intuition is that a sentence such as Revenue $> X$ may be supported by one branch and refuted by another. Following [17], a sentence is true (false) in an inconsistent model iff it is true (false) in some consistent contraction of that model. In our setting, "contractions" are the per-branch classical evaluations, and we sketch how to model that KPI sentence in a small logical language.

Language. Fix a quantifier-free KPI language $\mathcal{L}_{\text{kpi}}$ consisting of: (i) constant symbols such as Revenue, (ii) a constant symbol $\bar{X}$ for each numeric threshold X of interest, and (iii) binary predicate symbols $>$ and $=$. We are interested in sentences of the form

$$\varphi_X \;:=\; (\text{Revenue} > \bar{X}).$$

Branch models. Each branch b induces a classical $\mathcal{L}_{\text{kpi}}$-structure $\mathcal{M}_b$ with domain $|\mathcal{M}_b| = \mathbb{R}$ and standard interpretations of comparison predicates:

$$>^{\mathcal{M}_b} = \{(a, c) \in \mathbb{R}^2 \mid a > c\}, \qquad =^{\mathcal{M}_b} = \{(a, c) \in \mathbb{R}^2 \mid a = c\}.$$

Each KPI is interpreted by evaluating the corresponding query on branch b; e.g., Revenue evaluates to a real number. Satisfaction is then classical:

$$\mathcal{M}_b \models (\text{Revenue} > \bar{X}) \iff \text{Revenue}^{\mathcal{M}_b} > X.$$

An inconsistent model via contractions. Rather than defining a single non-classical numeric structure directly, we model the lakehouse state by the set of its admissible *consistent contractions*. Formally, let $\mathcal{M}$ be an "inconsistent" lakehouse whose contractions are the per-branch classical models:

$$\text{Con}(\mathcal{M}) \;:=\; \{\mathcal{M}_b \mid b \in B\}.$$

Intuitively, each $\mathcal{M}_b$ corresponds to one way of "clearing" the disagreement by committing to the values observed on a particular branch.

Subvaluational truth conditions. For any sentence ψ of $\mathcal{L}_{\text{kpi}}$, define truth and falsity in the lakehouse $\mathcal{M}$ by quantification over contractions:

$$\mathcal{M} \vDash^{+} \psi \iff^{\text{def}} \exists \mathcal{N} \in \text{Con}(\mathcal{M})\ (\mathcal{N} \models \psi)$$

$$\mathcal{M} \vDash^{-} \psi \iff^{\text{def}} \exists \mathcal{N} \in \text{Con}(\mathcal{M})\ (\mathcal{N} \not\models \psi)$$

Thus a sentence can be both true and false in $\mathcal{M}$ whenever different branches disagree. We call ψ settled true when $\mathcal{M} \vDash^{+} \psi$ and not $\mathcal{M} \vDash^{-} \psi$, equivalently when every contraction satisfies ψ; settled false is defined dually.

Worked examples. Fix $X = 100$ and consider the KPI sentence $\varphi := (\text{Revenue} > \bar{100})$.

(Glut.) Suppose there are two branches $a, b \in B$ such that

$$\text{Revenue}^{\mathcal{M}_a} = 120, \qquad \text{Revenue}^{\mathcal{M}_b} = 80.$$

Then $\mathcal{M}_a \models \varphi$ while $\mathcal{M}_b \not\models \varphi$.

By the subvaluational truth conditions,

$$\mathcal{M} \vDash^{+} \varphi \quad \text{and} \quad \mathcal{M} \vDash^{-} \varphi,$$

so φ is both true and false in $\mathcal{M}$: a truth glut driven by branch disagreement.

(Settled truth.) Reuse the same two branches $a, b \in B$, but let

$$\varphi' \;:=\; (\text{Revenue} > \bar{50}).$$

Then $\mathcal{M}_a \models \varphi'$ and $\mathcal{M}_b \models \varphi'$ since $120 > 50$ and $80 > 50$. Consequently,

$$\mathcal{M} \vDash^{+} \varphi' \quad \text{and not} \quad \mathcal{M} \vDash^{-} \varphi'.$$

Thus φ' is settled true across the lakehouse.

Table 2: Median latency (ms) by engine and branch count.

Q	Engine	1	2	4	8	16	32	50
Q1	Ad hoc	**1**	**4**	**5**	**11**	**21**	**41**	**62**
Q1	Native	3	13	15	25	44	88	136
Q3	Ad hoc	**288**	327	732	1390	2537	5125	7856
Q3	Native	366	**389**	**392**	**412**	**461**	**562**	**657**
Q4	Ad hoc	4	6	9	19	34	67	103
Q4	Native	**4**	**5**	**8**	**9**	**11**	**10**	**11**

B Further experiments

Table 2 gives the full data behind Section 4. The workload uses 54 branch variants of the predictions table, each with 2.5M rows, and a shared 3M-row dimension table as local Parquet files, with $B \in \{1, 2, 4, 8, 16, 32, 50\}$. Each configuration is run 5 times after 1 warmup; we report median wall-clock time on an Apple M3 machine with 36 GB RAM.