Polars CI Case Study: Computational Identity on Dataframe Query Plans

1 Overview

We apply Computational Identity (CI) to Polars, a dataframe library with a lazy evaluation mode that produces explicit query plans. This study follows the IronClaw pattern: three scenarios that demonstrate a flaw (silent optimizer drift), its detection (CI divergence), and the fix (version-tagged identity).

We test two Polars versions: **0.20.31** (old optimizer) and **1.38.1** (current optimizer).

2 Method

CI is computed as:

$$\text{CI}(e) = \text{SHA-256}(\texttt{polars:}\textit{version}\texttt{:}\text{plan}(e))$$

where plan(e) is the string returned by Polars' `LazyFrame.explain()`, which renders the optimized logical plan.

Unversioned CI omits the version prefix, exposing drift. Versioned CI includes it, making drift explicit by construction.

3 Scenario 1: Syntax Equivalence

Five pairs of semantically equivalent Polars expressions, written with different surface syntax:

Table 1: Syntax equivalence collapse rates by Polars version.

Test	**v0.20**	**v1.38**
Filter-then-group vs group-then-filter	DIFF	**SAME**
Predicate reorder (commutative AND)	DIFF	DIFF
Filter+select vs select+filter	SAME	SAME
`with_columns`+select vs direct select	DIFF	DIFF
Sort default vs explicit ascending	SAME	SAME
Collapse rate	2/5	**3/5**

Findings. Polars v1.38 gained one equivalence over v0.20: predicate pushdown through aggregation now produces identical plans for filter-then-group and group-then-filter. CI captures this improvement automatically—no rule changes needed.

Two boundary cases persist across both versions:

- **Predicate reorder**: Polars does not canonicalize the order of conjuncts in AND predicates. $(a > 900) \wedge (b = \texttt{"tech"})$ and $(b = \texttt{"tech"}) \wedge (a > 900)$ produce structurally different plans.
- `with_columns` **vs** `select`: These produce different plan node types (WITH_COLUMNS+π vs SELECT), an intrinsic structural boundary analogous to the SQL DISTINCT-vs-GROUP BY boundary.

4 Scenario 2: Version Drift

Five identical queries executed on both Polars versions. All five produce different plans:

Table 2: Cross-version plan drift. All queries drift.

Query	**Unversioned CI**	**Versioned CI**
filter+select	DRIFTED	Different (by design)
group+agg	DRIFTED	Different (by design)
predicate AND	DRIFTED	Different (by design)
sort+select	DRIFTED	Different (by design)
with_columns	DRIFTED	Different (by design)
Drift rate	**5/5**	—

The flaw. Every query produces a different optimized plan between v0.20 and v1.38. An unversioned CI (hash of plan string alone) would silently invalidate a computation cache built on one version when the system upgrades to another. No error, no signal—the cache keys simply stop matching, and the system silently recomputes everything. Worse: if the plan format changes in a way that happens to collide (different plans, same string), the cache returns stale results.

The fix. Versioned CI tags the hash input with the Polars version: $\text{CI}_{L,v}(e) = H(\texttt{polars:}v\texttt{:}\text{plan}(e))$. This makes drift explicit by construction. Same version, same query $\Rightarrow$ same CI. Different version $\Rightarrow$ different CI, guaranteed.

This is not a workaround—it is the correct formalization. The Polars optimizer is part of the language definition L. Changing the optimizer changes L, which changes CI. The version tag encodes this dependency (Property 3: Environment Independence; Limitation L2: Optimizer Version Dependence).

5 Scenario 3: Boundary Cases

Two structural boundaries are consistent across both versions:

- **Join commutativity**: `A.join(B)` and `B.join(A)` produce different plans (left/right sides are not reordered). The join is semantically commutative but structurally asymmetric—the same boundary as SQL's LEFT/RIGHT JOIN case.
- `with_columns` **vs** `select`: Different plan node types for equivalent computations. Analogous to SQL's DISTINCT vs GROUP BY boundary.

These are honest scope boundaries, not failures. CI captures structural equivalence, not semantic equivalence.

6 Comparison with IronClaw (SQL)

Table 3: IronClaw pattern across two domains.

Aspect	**SQL (IronClaw)**	**Polars**
Platform	DataFusion	Polars (Python)
Plan source	LogicalPlan	LazyFrame.explain()
Versions tested	DF 38 vs 44	Polars 0.20 vs 1.38
Drift rate	3/3 scenarios	5/5 queries
Equivalences	22/22 TPC-H classes	3/5 syntax pairs
Boundary cases	5 (adversarial suite)	2 (join, node type)
Fix	Version-tagged CI	Version-tagged CI

The same CI framework—canonicalize, plan, serialize, hash—applies to both SQL query engines and dataframe libraries. The drift pattern is identical: optimizer upgrades silently change plans, and only CI makes this visible.

7 Reproducibility

All code is in `polars_ci_study.py`. To reproduce:

```
pip install polars==1.38.1
python3 polars_ci_study.py

python3 -m venv /tmp/old && /tmp/old/bin/pip install polars==0.20.31
/tmp/old/bin/python3 polars_ci_study.py

python3 compare_versions.py
```