SPIP: Distinguish Row-Order-Independent Nondeterministic Expressions in Catalyst

- **Author:** Saurabh Maurya
- **Shepherd:** TBD (Spark PMC)
- **JIRA:** SPARK-62199 · **PR #1a:** PR #60776 · **PR #1b (WIP Draft):** PR #60778
- **Target Version:** Apache Spark 5.0.0

TLDR

Catalyst blocks filter, dynamic partition pruning (DPP), semi-join, and DSv2 offset pushdowns across nondeterministic projections because Expression.deterministic conflates repeatability with row-order, row-subset, and partition-context dependence.

Opt-in Expression.rowOrderIndependent inherits deterministic (false for all nondeterministic expressions), unlocking those pushdowns while keeping deterministic = false and stateful expressions unchanged.

Expensive row-order-independent expressions (such as random-IV aes_encrypt or external/AI RPC UDFs) skip filtered-out rows and stop blocking partition pruning, DPP, and scan pushdown with no measurable overhead for cheap UDFs (see Benchmarks), without weakening rand(), multi-ref CTE inlining, ANSI order, or SPARK-59726.

Big Questions for Reviewers

1. **Contract & name:** Does this per-row value-only contract (opt-in Expression.rowOrderIndependent vs extending Expression.stateful) fit Catalyst?
2. **Sequencing:** Can the Catalyst-internal patch (no public API) land first with aes_encrypt (PR #58867 / PR #60178) as proposed first in-tree caller (2-line follow-up) while public APIs follow the SPIP vote?
3. **Public UDF naming:** On UserDefinedFunction, prefer asRowOrderIndependent() or asRowOrderIndependentNondeterministic()?

Q1: What are you trying to do?

- **Contract:** An expression has rowOrderIndependent = true when its output distribution depends solely on that row's input values — never on row order, row count/subset, or task context (spark_partition_id(), input_file_name()). Unseeded rand() and uuid() stay false because their per-partition PRNG stream depends on row position. Below, a disjoint predicate references no nondeterministic output attributes.
- **Vs RDD DeterministicLevel:** Spark Core's DeterministicLevel governs stage re-

execution; rowOrderIndependent is a Catalyst expression property governing algebraic pushdowns while keeping deterministic = false.

- **Vs rejected PR #44460:** PR #44460 (SPARK-46487) relaxed pushdowns globally, breaking rand(seed) and monotonically_increasing_id(). Inheriting deterministic keeps all built-ins and .asNondeterministic() UDFs blocked unless opted in.

Tier	deterministic	rowOrderIndependent	Examples	Safe Rewrites	Blocked Rewrites
Deterministic	true	true	+, upper, pure UDFs	All	None
Row-Order-Independent	false	true	Opt-in StaticInvoke (proposed: AesEncrypt), UDFs, DSv2 ScalarFunction	Disjoint filters, CTE ref filters, DPP/Bloom, semi/anti-join, DSv2 offset, count == 1 collapse	Fold, CSE, multi-ref CTE inlining, scan nondeterministic filters, limit-projection pushdown, PolicyEnforcement
Order- / Context-Dependent	false	false	rand, uuid, shuffle, monotonically_increasing_id, spark_partition_id, .asNondeterministic()	No new rewrites (existing SPARK-13473 barriers intact)	All reorderings & duplications

Q2: What problem is this NOT designed to solve?

- **Built-in defaults:** rand(), randn(), uuid(), shuffle(), monotonically_increasing_id(), spark_partition_id(), input_file_*(), current_batch_timestamp() stay false.
- **Existing semantics:** Expression.deterministic, Expression.stateful, and legacy .asNondeterministic() UDFs stay unchanged.
- **Repeatability or retries:** No folding, CSE, f(x) = f(x) dedup, or exact-once retries (SPARK-61672).
- **CTEs & scan filters:** Multi-ref InlineCTE (SPARK-36447, 1x via ReusedExchange), constraints, and scan nondeterministic filters (SPARK-58207) stay deterministic-only; outer disjoint CTE reference predicates (SPARK-59434, PushdownPredicatesAndPruneColumnsForCTEDef) push through row-order-independent CTE projections to the scan with 1x execution.

- **Security barriers:** Relaxed rules never cross SPARK-59726 (proposal) PolicyEnforcement; nondeterministic expressions never qualify as LEAKPROOF.

Q3: How is it done today, and what are the limits?

SPARK-13473 / SPARK-20246 block pushdowns across nondeterministic Projects/Aggregates because deterministic conflates repeatability with order/state dependence (SPARK-27969, dev@spark). Three efforts hit this:

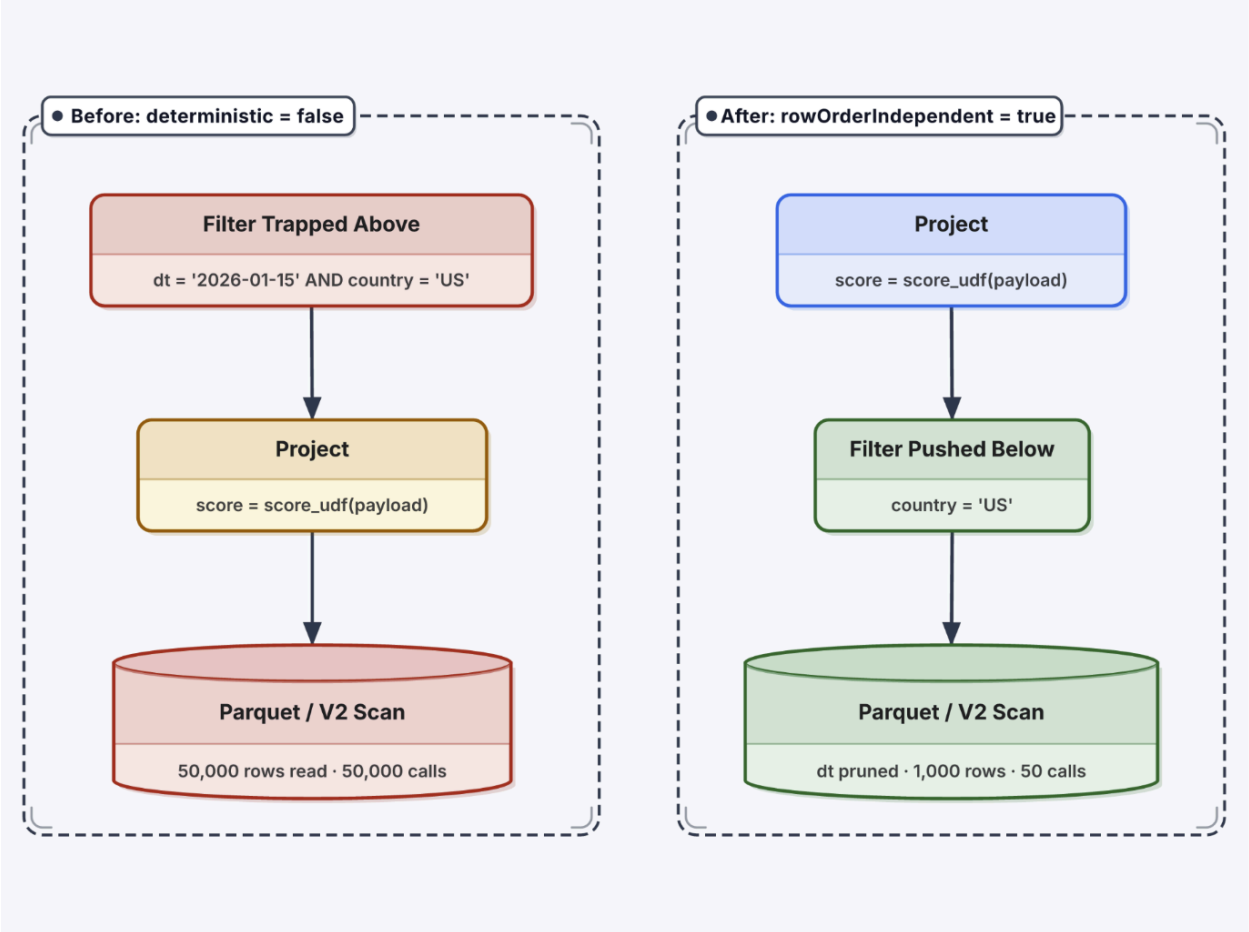
- **Random-IV aes_encrypt:** Making SecureRandom aes_encrypt nondeterministic triggered pushdown-regression findings on PR #58867 (SPARK-59768; automated review, author ack) and PR #60178 (SPARK-60934).
- **AI & external UDFs:** On Native AI (SPARK-61672, SPARK-61404 / PR #60311; thread), Daniel Tenedorio (comment) and Norio Akagi (comment) flagged binary isDeterministic forcing duplication or blocked pushdowns.
- **Views & CTEs:** Manual filter-before-project fails in TEMP VIEWs, CTEs, BI SQL, or DataFrame helpers.

Example 1: View Partition Pruning

```
CREATE VIEW scored AS SELECT id, country, dt, score_udf(payload) AS score FROM events;
SELECT id, score FROM scored WHERE dt = '2026-01-15' AND country = 'US';

-- Before: filter trapped, 50-partition scan
Project [id, score]
+- Filter (dt = '2026-01-15' AND country = 'US')
   +- Project [id, country, dt, score_udf(payload) AS score]
      +- FileScan events PartitionFilters: []

-- After: partition pruned, country filter pushed to Parquet scan, outer Project
collapsed
Project [id, score_udf(payload) AS score]
+- Filter (country = 'US')
   +- FileScan events PartitionFilters: [dt = '2026-01-15'], PushedFilters: [country =
'US']
```



Example 2: Star-Schema DPP and Post-Projection Filter

```
SELECT * FROM (SELECT dt, mask_udf(ssn) AS enc FROM fact) f
JOIN dim d ON f.dt = d.dt WHERE f.enc IS NOT NULL AND d.region = 'US';

-- Probe side before: DPP trapped, dropped by CleanupDynamicPruningFilters
Filter (enc IS NOT NULL AND dynamicpruning(dt))
+- Project [dt, mask_udf(ssn) AS enc]
   +- FileScan fact PartitionFilters: []

-- Probe side after: enc filter stays post-projection; DPP reaches scan
Filter (enc IS NOT NULL)
+- Project [dt, mask_udf(ssn) AS enc]
   +- FileScan fact PartitionFilters: [dynamicpruning(dt)]
```

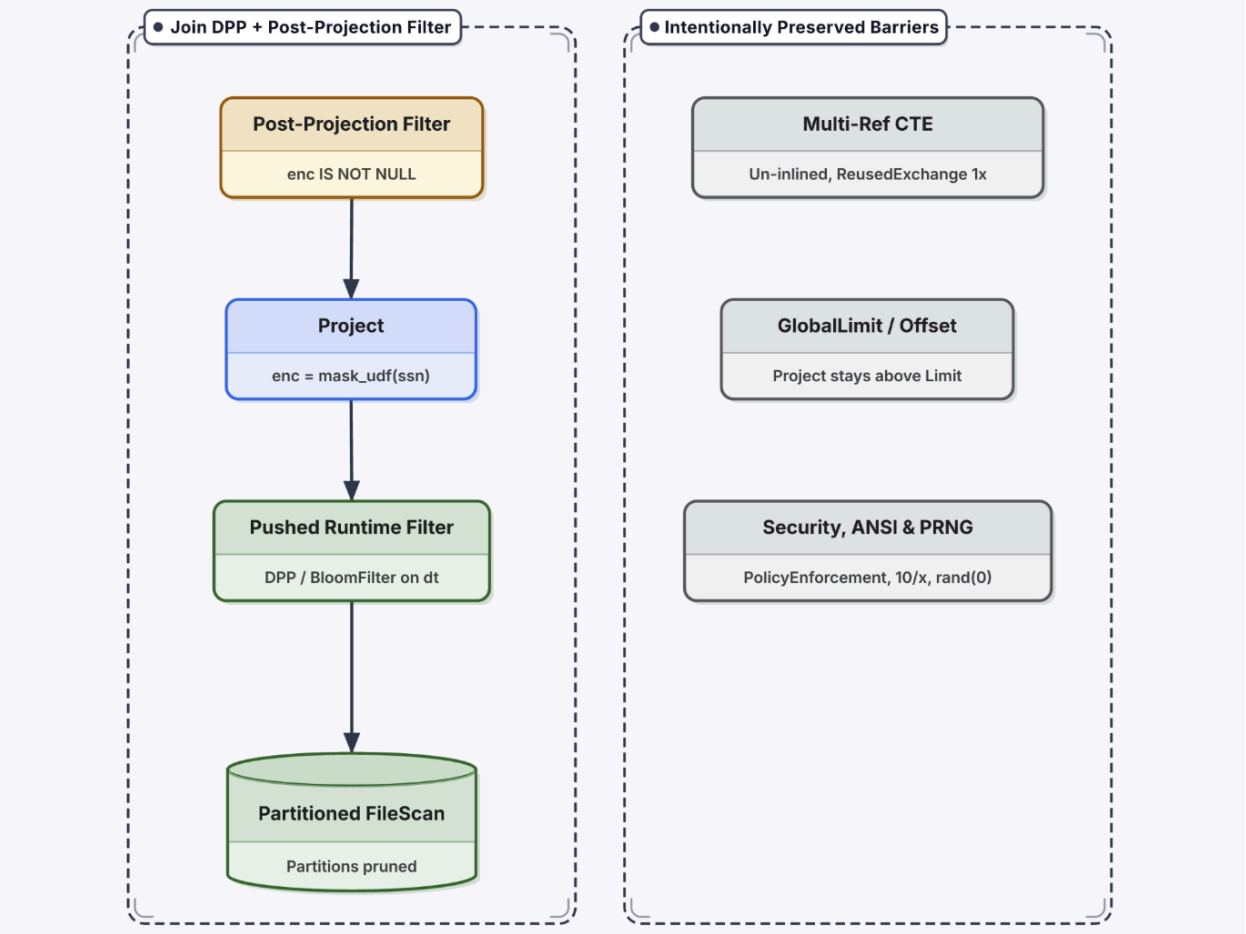
Example 3: Safety Boundaries (Intentionally Blocked)

```
-- (a) Multi-ref CTE un-inlined: 1x calls via ReusedExchange
WithCTE
:- CTERelationDef s [Project score_udf(payload) AS score]
+- Join (CTERelationRef s, CTERelationRef s)

-- (b) Project stays above GlobalLimit: 5 calls, never 5 x numPartitions
Project [id, mask_udf(v) AS enc]
+- GlobalLimit 5
   +- LocalLimit 5

-- (c) Post-projection ANSI guard: enc IS NOT NULL guards 10 / x > 1 (avoids
DIVIDE_BY_ZERO)
Filter (enc IS NOT NULL AND 10 / x > 1)
+- Project [x, mask_udf(v) AS enc]

-- (d) Seeded rand(0) blocked (deterministic = false, rowOrderIndependent = false)
Filter (a > 10)
+- Project [a, rand(0) AS r]
```



Cross-Engine Precedent

Engine	Model	Separates State/Order?	Pushes Unrelated?	Source
Apache Spark (today)	Single deterministic flag	No (SPARK-13473)	No	Expression.scala
ClickHouse	isDeterministic + isStateful	Yes	Yes (!isStateful)	IFunction.h
Apache Doris	Volatile + NoneMovableFunction	Yes	Yes (!NoneMovable)	PushDownFilterThroughProject.java
PostgreSQL	VOLATILE+LEAKPR	Hive: yes; PG:	Yes (PG blanket; Spark	allpaths.c,

& Hive	OOF / @UDFType	no	protects seeded PRNGs & monotonically_increasing_id)	UDFType.java

Q4: What is new in your approach?

```
abstract class Expression extends TreeNode[Expression] {
  lazy val deterministic: Boolean = children.forall(_.deterministic)
  // lazy val in prototype; can become def with if (deterministic) true fast-path (cf.
PR #60469)
  lazy val rowOrderIndependent: Boolean =
    if (Expression.overridesDeterministic(getClass)) deterministic
    else children.forall(_.rowOrderIndependent)
}
// Nondeterministic hits override branch & sets false; StaticInvoke adds
isRowOrderIndependent = false.
// sql/api UserDefinedFunction (@since 5.0.0; concrete default preserves binary
compatibility):
def rowOrderIndependent: Boolean = deterministic
def asRowOrderIndependentNondeterministic(): UserDefinedFunction =
asNondeterministic()
// DSv2 ScalarFunction (@since 5.0.0, not BoundFunction):
default boolean isRowOrderIndependent() { return isDeterministic(); }
```

- **Override safety:** Subclasses overriding deterministic (Hive UDFs, ScalaUDAF, Uniform, third-party) get rowOrderIndependent = deterministic via cached ClassValue catching LinkageError conservatively (CodeGenerator.leavesGenCodeToExpression, definition; ClassLoader-tested); default-inheriting subclasses propagate from children. Invoke, StaticInvoke, and PlanExpression compose explicitly; a def fast-path (PR #60469 def isTotal) avoids per-Expression fields.
- **Row-count monotonicity:** Opted-in projections never process extra rows: pushing LIMIT/OFFSET below a projection (V2ScanRelationPushDown) shrinks input safely, whereas pushing projections below GlobalLimit/Offset (PushProjectionThroughLimitAndOffset) fans out n to n x numPartitions and stays blocked.

Rule Group	Guard Change	Safety Invariant
Filter	PushPredicateThroughNonJoin relaxes	Disjoint predicates, ANSI barriers, DPP/Bloom

pushdown	Project and grouped Aggregate (PR #1a), plus BatchEvalPython/ArrowEvalPython (PR #1b)	bypass; Distinct/Generate/Pivot/Expand/Window stay deterministic
Semi/Anti & scan	PushDownLeftSemiAntiJoin allows row-order-independent Project and grouped Aggregate	Disjoint predicates, join-scoped aliasMap; OperationHelper (patterns.scala) stays deterministic-only to match StreamingV2ScanPushDown
Limit & Offset	V2ScanRelationPushDown.pushDownOffset relaxed; PushProjectionThroughLimitAndOffset untouched	LIMIT/OFFSET below projection shrinks input; never n to n x numPartitions
Collapse & guards	CollapseProject.mergeProjectExpressions inlines single-ref non-lambda row-order-independent expressions	CollapseProject.canCollapseExpressions stays unchanged (deterministic-only for CombineUnions, patterns.scala, V2ScanRelationPushDown); ignores alwaysInline/ isCheap (PR Breakdown)

Q5: Who cares?

- **SQL encryption:** (SPARK-59768, SPARK-60934) Random-IV aes_encrypt with pruning, DPP, and semi-joins.
- **AI & UDFs:** (SPARK-61672, SPARK-61404) Remote LLM/gRPC/REST UDFs behind views/ CTEs with pruning and 1x calls.
- **DSv2 & workers:** (SPARK-55278) Connectors mis-declare isDeterministic = true for pushdown; opt-in rowOrderIndependent replaces that workaround.
- **Ecosystem engines:** Delta Lake, Iceberg, Gluten, and Comet still see deterministic = false.

Q6: What are the risks?

Risk	Mitigation
Third-party subclasses	ClassValue catches LinkageError conservatively; deterministic-overriding classes get rowOrderIndependent = deterministic unless overridden.
Call duplication	Single-ref non-higher-order inlining, no nondeterministic alias substitution, same-resultId PythonUDF dedup, Python UDF cache invalidation.
ANSI & DPP/	Written-order splitPushable blocks fallible predicates while passing total-key DPP/Bloom

Bloom	(SPARK-60602, SPARK-61663).
Limit fan-out	PushProjectionThroughLimitAndOffset stays deterministic-only; no new LimitPushDown arm; existing LimitPushDown rules only move limits below projections (shrinking input safely).
Rollout & Connect	Gated behind an internal kill-switch SQLConf (spark.sql.optimizer.rowOrderIndependentNondeterministicPushdown.enabled, default true); existing built-ins stay false; Connect proto field 8 is optional.

Q7: How long will it take?

Prior Catalyst-internal properties (SPARK-47672 expensive, SPARK-61663 isTotal) landed via JIRA + PR. Pending PMC agreement, PR #1a (Catalyst-internal; aes_encrypt proposed first in-tree caller, 2-line follow-up) unblocks PR #58867 / PR #60178 while PR #1b goes through the SPIP vote.

Phase	Scope	Sequencing
PR #1a (Catalyst-internal)	rowOrderIndependent, StaticInvoke flag, core optimizer rules	PR #60776 (ready); unblocks PR #58867/#60178 (PR Breakdown)
PR #1b (Public API)	Scala/Java/PySpark UDFs, DSv2 ScalarFunction, Connect field 8, UDAF guards	WIP Draft PR #60778; MiMa/proto clean; SPIP vote (PR Breakdown)
Follow-ups (1–2 weeks)	AesEncrypt 2-line opt-in (proposed first in-tree caller), build-side DPP/cache, SPARK-55278	Independent sub-tasks

Q8: How will we measure success?

Criterion	Target / Measured Result
Optimizer & API gates	All targeted Catalyst, SQL Core, Connect, and PySpark parity suites pass; proto sync, linters, and MiMa (vs Spark 4.2.0) are clean (details in PR Breakdown).
Pushdown & pruning	Disjoint filters reach Parquet/DSv2 scans (partition pruning & DPP active) and skip filtered-out rows across Scala, DSv2, and PySpark UDFs (Benchmarks)

External / RPC UDFs	Avoids evaluating external/RPC UDFs on filtered-out rows; no measurable overhead at 0 ms (Benchmarks)
CTE & compatibility	1x multi-ref CTE (100% ReusedExchange; ref predicates push to scan), 0 MiMa issues vs 4.2.0, rand(0) parity

Alternatives Considered

Comparison across candidate approaches:

Approach	Pushdown	No Call Dup.	Legacy Safety	Generality	Verdict
rowOrderIndependent + ClassValue	Full	Preserved (deterministic = false)	Safe (inherits deterministic)	Built-ins, UDFs, DSv2, AI, Connect	Recommended
deterministic = true (+ expensive)	Full	Broken (fold, CSE, CTE, MERGE)	Unsafe (silent bugs: SPARK-47672, SPARK-59693)	Scalar UDFs only	Rejected
Invert Expression.stateful	Partial	Preserved (deterministic = false)	Unsafe (ScalaUDF is stateful; SparkPartitionID is not)	Wrong abstraction	Rejected
Blanket relaxation (PR #44460)	Full	Preserved (deterministic = false)	Unsafe (breaks rand, uuid, monotonically_increasing_id)	All expressions	Rejected
AI-only Project operator	AI functions only	Preserved (dedicated rule)	Safe (opt-in)	Narrow (misses AesEncrypt and general UDFs)	Rejected

PR Breakdown, Test Invariants, and Verification Gates

Delivery splits into two sequential PRs on apache/spark master (5.0.0-SNAPSHOT, rebased on 634e3261; PR #60776 & WIP Draft PR #60778) under one Umbrella JIRA: Catalyst-internal patch PR #1a (no public API, consistent with SPARK-47672 / SPARK-61663; pending PMC sequencing agreement, unblocks random-IV aes_encrypt as proposed first in-tree caller via a 2-line follow-up for PR #58867 and PR #60178) followed by SPIP-scoped public API and Spark Connect PR #1b.

Upstream PR #58867 (SPARK-59768) already rewrites NondeterministicExpressionCollection and PullOutNondeterministic. Leaving both files untouched avoids conflicts with PR #58867; no migration-guide entry and no legacy GROUP BY/ORDER BY change.

PR #1a: Catalyst-Internal Patch (No Public API)

Module	File	Change
sql/ catalyst	Expression.scala	Add rowOrderIndependent inheriting deterministic (false when nondeterministic) via cached ClassValue override check catching LinkageError conservatively (matching CodeGenerator.leavesGenCodeToExpression); Nondeterministic explicitly sets false
sql/ catalyst	objects/ objects.scala	Add defaulted StaticInvoke.isRowOrderIndependent = false; override on StaticInvoke/Invoke; adjust positional patterns in stringExpressions.scala and CharVarcharUtils.scala
sql/ catalyst	subquery.scala	Override PlanExpression.rowOrderIndependent = children.forall(_.rowOrderIndependent) && plan.deterministic
sql/ catalyst	SQLConf.scala	Add internal SQLConf rowOrderIndependentNondeterministicPushdown.enabled (.internal(), ConfigBindingPolicy.NOT_APPLICABLE, 5.0.0, default true)
sql/ catalyst	Optimizer.scala	Relax PushPredicateThroughNonJoin (Project and grouped Aggregate; ANSI/DPP); CollapseProject.mergeProjectExpressions handles single-ref non-lambda row-order-independent inlining while CollapseProject.canCollapseExpressions remains unchanged (preserving deterministic-only behavior for CombineUnions, patterns.scala, V2ScanRelationPushDown)
sql/	planning/	Unchanged (stays deterministic-only to preserve

catalyst	patterns.scala	StreamingV2ScanPushDown mirror)
sql/ catalyst	PushDownLeftSemi AntiJoin.scala	Relax Project (join-ref aliasMap scoping) and grouped Aggregate for disjoint predicates
sql/ catalyst	util/ V2ExpressionBuilder .scala	Adjust 10-arg StaticInvoke pattern match only (no behavior change; callers check determinism)
sql/core	V2ScanRelationPush Down.scala	Relax pushDownOffset across row-order-independent Project (pushDownLimit already traverses Project; PushProjectionThroughLimitAndOffset untouched)
sql/ catalyst & sql/ core tests	Unit & integration suites	Catalyst/Core unit, ClassLoader LinkageError, CTE ref-predicate pushdown, ANSI/DPP, multi-partition limit-guard, and kill-switch suites (upstream SPARK-61482 already fixes ExtractPythonUDFs dedup on master)

PR #1b: Public UDF / DSv2 APIs & Spark Connect Parity

Module	File	Change
common/ utils	error-conditions.json	Add UNSUPPORTED_FEATURE.ROW_ORDER_INDEPENDENT_AGGREGATOR
sql/api	UserDefinedFunction.scala	Add @since 5.0.0 rowOrderIndependent and opt-in builder (prototype: asRowOrderIndependentNondeterministic(); naming open per SPIP) with MiMa defaults; fix chaining reset; reject UserDefinedAggregator
sql/ catalyst	ScalarFunction.java, ScalaUDF, PythonUDF, ApplyFunctionExpression, objects.scala	Add @since 5.0.0 ScalarFunction.isRowOrderIndependent() (not BoundFunction); wire udfRowOrderIndependent; relax PushPredicateThroughNonJoin for BatchEvalPython/ArrowEvalPython
sql/core	UserDefinedFunctionUtils, ExtractPythonUDFFromLambda, UserDefinedPythonFunction	Forward udfRowOrderIndependent; reject all 9 grouped/incremental/window aggregate PythonEvalTypes
python/	udf.py, _typing.pyi, udf.rst	Expose Python API (incl. scalar @pandas_udf),

pyspark & docs		invalidate Python UDF caches on builder mutation, preserve bufferSchema, reject aggregates
sql/ connect & pyspark	expressions.proto, UdfToProtoUtils, SparkConnectPlanner, connect/ udf.py	Add optional bool row_order_independent = 8 on CommonInlineUserDefinedFunction; sparse encode; reject UDAFs
sql/core, connect, py tests	UDFSuite, DataSourceV2FunctionSuite, Connect & PySpark parity suites	1x AtomicInteger/Accumulator proofs, Classic/ Connect parity, raw wire-bytes tests

Follow-Up PRs

Follow-Up	Scope	Notes
AesEncrypt opt-in	2-line rowOrderIndependent + StaticInvoke flag	Proposed first in-tree caller (2-line follow-up after PR #58867 / PR #60178, or folded into PR #1a if PMC prefers)
Build-side DPP & cache	Relax unused row-order-independent pruning on DPP build side & cache lineage	Separate broadcast reuse semantics
Language-Agnostic UDFs	Propagate row_order_independent in SPARK-55278 worker specs	Unset inherits deterministic

Grouped Key Test Invariants

Invariant Group	PR	Suites	Verified Guarantee
Built-in & third-party defaults	#1a/ #1b	RowOrderIndependentExpressionSuite	FunctionRegistry sweep, custom subclasses, ClassLoader LinkageError conservative fallback, rand/ uuid/monotonically_increasing_id, StaticInvoke(Rand) stay false
Filter, Parquet & patterns	#1a/ #1b	FilterPushdownSuite, UDFSuite, DataSourceV2FunctionS	Disjoint predicates push to Parquet/DSv2 and PhysicalOperation/ScanOperation; 1x AtomicInteger calls

		uite	
ANSI barriers & DPP/Bloom	#1a	FilterPushdownSuite, RowOrderIndependentPushdownSuite	Post-projection RaiseError/Divide/Cast/throwing UDF stay above Project; DPP & Bloom pass post-projection filters on row-order-independent outputs
Semi/Anti, Offset & Limit	#1a	LeftSemiAntiJoinPushDownSuite, DataSourceV2StrategySuite, PushProjectionThroughLimitAndOffsetSuite, RowOrderIndependentPushdownSuite	Multi-arg row-order-independent projections push semi/anti-joins and DSv2 offset; Project stays above GlobalLimit/LocalLimit/Offset (multi-partition LIMIT 5 makes exactly 5 calls)
CollapseProject non-dup	#1a/ #1b	CollapseProjectSuite	count == 1 collapses; count == 0 prunes; count > 1 blocked (even alwaysInline/isCheap); higher-order functions blocked
PySpark & ExtractPythonUDFs	#1b	ExtractPythonUDFsSuite, test_udf.py, test_parity_udf.py	Filter pushes below Batch/ArrowEvalPython; upstream SPARK-61482 same-resultId dedup stays intact
CTE, constraints & V2Expr	#1a	InlineCTESuite, InferFiltersFromConstraintsSuite, PruneFiltersSuite	Multi-ref CTE stays un-inlined logically in PR #1a (1x via ReusedExchange verified in Benchmarks/PR #1b) while outer disjoint ref predicates (PushdownPredicatesAndPruneColumnsForCTEDef) push through row-order-independent CTE projections to scan (blocked for rand(0) / kill-switch); no constraint inference/pruning
API, UDAF, wire & flag	#1a/ #1b	SparkThrowableSuite, Connect suites, UDFSuite	Chaining reset, 9 aggregate eval types/UDAFs throw, tag 0x40 + unknown tag 99 pass, kill-switch disables all relaxed rewrites

Verified Gate Matrix

Gate	Scope	Verified Result
PR #1a Targeted Suites & Lint	20 files (+2351/-28; 10 prod files +242/-24, 7 core logic files)	20 targeted Catalyst suites: 536/536 active passed (1 pre-existing skipped; RowOrderIndependentExpressionSuite 10/10, InlineCTESuite 12/12); 13 targeted SQL Core suites: 545/545 active passed (6 pre-existing skipped; RowOrderIndependentPushdownSuite 4/4, DataSourceV2StrategySuite 60/60, upstream

		ExtractPythonUDFsSuite 16/16, InjectRuntimeFilterSuite 67/67, DynamicPartitionPruningV1/V2 (AEOff+AEOn) 188/188 active); scalastyle 1,624 files (0 errors/warnings)
PR #1b Cross-Module, PySpark & Lint	33 files (+1533/-61)	18 cross-module & Connect suites (607 active passed, 1 pre-existing skipped: SparkThrowableSuite 38/38, UDFSuite 85/85, DataSourceV2FunctionSuite 48/48, ExtractPythonUDFsSuite 17/17, PythonUDFSuite 20/20, BatchEvalPythonExecSuite 17/17, Connect 4 suites 73/73); PySpark Classic test_udf (98 passed) and Connect test_parity_udf (84 active passed); scalastyle, scalafmt, Checkstyle, ruff, and mypy clean (0 issues)
API & Wire Compatibility	sql/api, sql/connect, common, Python	Connect proto sync (dev/check-protos.py SUCCESS); ./dev/mima vs Spark 4.2.0 (0 binary compatibility issues)

Live PRs

- **PR #1a (Catalyst-internal):** apache/spark#60776 (base 634e3261, single clean commit, Ready for Review) · **PR #1b (Public APIs, WIP Draft):** apache/spark#60778 (stacked on #60776)

Local Parquet + External-UDF Benchmarks & Captured Physical Plans

Headline Findings

- **Plan & call-count verification (0.1% selectivity):** Pushing view filters into FileSourceScanExec prunes 49 of 50 Parquet partitions (50,000 to 1,000 scanned rows; 50,020 to 1,020 in B2) and evaluates Scala, DSv2, and PySpark UDFs only on the 50 matching rows instead of all 50,000.
- **Latency check:** With a 2 ms simulated per-row RPC, end-to-end time drops from ~28 s to 0.2–0.6 s (~47x–153x, tracking filter selectivity); at 0 ms, there is no measurable regression.
- **1x multi-ref CTE reuse:** Keeping deterministic = false leaves refCount == 2 CTEs un-inlined while pushing inner predicates to Parquet, yielding 50 calls (1x) and 50/50 (100%) stochastic agreement via ReusedExchange.

Environment & Variance

- **Build & host:** Apache Spark 5.0.0-SNAPSHOT prototype build (identical base commit for both arms), local[4], AQE on, 8 shuffle partitions, OpenJDK 21.0.12.1 (-Xmx4g), Python 3.13.15 (PyArrow 26.0.1), 24-vCPU x86_64 Linux (94 GiB RAM).
- **Dataset & filter:** Synthetic 50,000-row Parquet events (50 dt partitions x 20 countries x 50 rows) + 20-row country_dim; WHERE dt = '2026-01-15' AND country = 'US' returns 50 rows (0.1%).
- **Counters & runs:** JVM AtomicLong (Scala/DSv2) and Spark Accumulators (PySpark) count calls; deduplicated FileSourceScanExec numOutputRows counts scans; 2 warmups / 3 runs at 0 ms, 1 warmup / 2 Baseline or 3 Proposed runs at 2 ms.
- **Variance note:** Observed min-max spreads are 3–264 ms across n = 2–3 runs due to local[4] scheduling, AQE, GC, and Python IPC; treat 0 ms ratios (*) as no-overhead checks and rely on zero-variance counts and 2 ms RPC speedups.

Measured Results Summary

All 24 rows produce 50 output rows (*0 ms latency ratios are indicative; B3 is Proposed-only to verify 1x ReusedExchange reuse, B5b is 0 ms only to verify Arrow IPC pushdown, DSv2-Static = static magic invoke, DSv2-Produce = produceResult).

Workload	Arm	Scanne	Calls	Avg ms	Min–Max ms	Calls /

		d				Speedup
B1 ScalaUDF	0ms Base	50,000	50,000	205.16	183.30–235.52	1.0x
B1 ScalaUDF	0ms Prop	1,000	50	115.09	108.21–123.55	1,000x, 1.78x*
B1 ScalaUDF	2ms Base	50,000	50,000	27,717.54	27,585.46–27,849.63	1.0x
B1 ScalaUDF	2ms Prop	1,000	50	200.66	198.28–202.12	1,000x, 138.13x
B2 View-Join	0ms Base	50,020	50,000	256.02	239.43–286.62	1.0x
B2 View-Join	0ms Prop	1,020	50	172.03	166.79–177.11	1,000x, 1.49x*
B2 View-Join	2ms Base	50,020	50,000	27,724.77	27,709.70–27,739.83	1.0x
B2 View-Join	2ms Prop	1,020	50	587.96	461.06–687.22	1,000x, 47.15x
B3 Multi-CTE	0ms Prop	1,000	50 (1x)	326.28	213.89–404.85	100% (50/50)
B3 Multi-CTE	2ms Prop	1,000	50 (1x)	305.07	292.79–319.39	100% (50/50)
B4a DSv2-Static	0ms Base	50,000	50,000	121.12	114.53–127.49	1.0x
B4a DSv2-Static	0ms Prop	1,000	50	83.90	71.08–106.54	1,000x, 1.44x*
B4a DSv2-Static	2ms Base	50,000	50,000	27,592.82	27,562.05–27,623.58	1.0x
B4a DSv2-Static	2ms Prop	1,000	50	182.24	175.96–191.20	1,000x, 151.41x
B4b DSv2-Produce	0ms Base	50,000	50,000	128.59	118.87–141.18	1.0x
B4b DSv2-Produce	0ms Prop	1,000	50	70.92	68.85–74.51	1,000x, 1.81x*
B4b DSv2-Produce	2ms Base	50,000	50,000	27,685.64	27,649.45–27,721.83	1.0x
B4b DSv2-Produce	2ms Prop	1,000	50	180.38	178.94–181.62	1,000x, 153.49x

B5a Py-Batch	0ms Base	50,000	50,000	358.16	327.43–381.03	1.0x
B5a Py-Batch	0ms Prop	1,000	50	232.81	209.98–255.11	1,000x, 1.54x*
B5a Py-Batch	2ms Base	50,000	50,000	27,981.54	27,955.30–28,007.78	1.0x
B5a Py-Batch	2ms Prop	1,000	50	263.69	247.80–287.29	1,000x, 106.12x
B5b Py-Arrow	0ms Base	50,000	50,000	216.76	207.98–230.85	1.0x
B5b Py-Arrow	0ms Prop	1,000	50	150.35	145.85–154.88	1,000x, 1.44x*

Captured Physical Plans

B1 Parquet View (ScalaUDF): Baseline vs Proposed

```
-- Baseline (.asNondeterministic):
*(1) Project [id#33, score#136]
+- *(1) Filter ((isnotnull(dt#36) AND isnotnull(country#34)) AND ((dt#36 = 2026-01-15)
AND (country#34 = US)))
   +- *(1) Project [id#33, country#34, dt#36, score_baseline(payload#35) AS score#136]
      +- FileScan parquet [id#33,country#34,payload#35,dt#36] DataFilters: [],
PartitionFilters: [], PushedFilters: []

-- Proposed (row-order-independent opt-in):
*(1) Project [id#33, score_proposed(payload#35) AS score#215]
+- *(1) Filter (isnotnull(country#34) AND (country#34 = US))
   +- FileScan parquet [id#33,country#34,payload#35,dt#36] DataFilters:
[isnotnull(country#34), (country#34 = US)], PartitionFilters: [isnotnull(dt#36),
(dt#36 = 2026-01-15)], PushedFilters: [IsNotNull(country), EqualTo(country,US)]
```

B3 Multi-Reference CTE: AQE ReusedExchange (1x Calls, 100% Score Match)

```
*(4) BroadcastHashJoin [id#783, score#786], [id#792, score#793], Inner, BuildRight
:- ShuffleQueryStage 0 -> Exchange RoundRobinPartitioning(8), [plan_id=2788]
:  +- *(1) Project [id#803, score_proposed(payload#805) AS score#807]
:     +- *(1) Filter ((isnotnull(country#804) AND (country#804 = US)) AND
isnotnull(id#803))
:        +- FileScan parquet [id#803,country#804,payload#805,dt#806] PartitionFilters:
[dt#806 = 2026-01-15], PushedFilters: [EqualTo(country,US), IsNotNull(id)]
+- BroadcastQueryStage 3 -> ShuffleQueryStage 2 -> ReusedExchange [id#808, score#812],
```

```
[plan_id=2788]
```

B5b PySpark @udf (ArrowEvalPython, Spark 5.0 Default): Proposed (50 Rows Cross IPC)

```
*(2) Project [id#0, pythonUDF0#208 AS score#206]
+- ArrowEvalPython [py_arrow_proposed(payload#2)#207], [id#0], [pythonUDF0#208], 101
   +- *(1) Project [id#0, payload#2]
      +- *(1) Filter (isnotnull(country#1) AND (country#1 = US))
         +- FileScan parquet [id#0,country#1,payload#2,dt#3] PartitionFilters: [dt#3 =
2026-01-15], PushedFilters: [IsNotNull(country), EqualTo(country,US)]
```

Local Reproduction

The Catalyst-internal implementation is live in PR #60776 (including RowOrderIndependentPushdownSuite, which verifies Parquet/DSv2 partition pruning, DPP, multi-partition LIMIT call counts, and ANSI barriers in-tree), and the stacked public UDF/DSv2/ Connect API prototype is live in WIP Draft PR #60778.

```

```