

Automated Operator Fusion for GPU-Accelerated SQL: Generation and Evolutionary Refinement

Corey Lammie*, Yotam Perlitz, Andrea Giovannini, Abdel Labbi

ADMS 2026, Boston, MA

August 31, 2026

IBM Research, Säumerstrasse 4, 8803 Rüschlikon, Switzerland

*corey.lammie@ibm.com

What This Talk Is About

- GPU SQL engines already accelerate relational operators with libraries such as cuDF;
- The remaining question is whether **operator boundaries** are the right GPU execution boundaries;
- We generate and refine **fused Triton kernels** for bounded Velox pipeline windows;
- **Correctness gates** come before performance measurement.

Claim: **Pipeline-granularity fusion** is a practical middle point between per-operator GPU lowering and whole-workload engine synthesis.

Motivation

The GPU Backend Pattern

- Existing engines keep the CPU-style operator stack;
- Supported operators are lowered to **GPU library calls**;
- Unsupported operators **fall back** to the ordinary execution path;
- This is maintainable and composable, but preserves many physical boundaries.

Velox

operator abstraction

cuDF

GPU operator backend

Triton

generated fused kernels

Where the Cost Comes From

Per-operator GPU execution

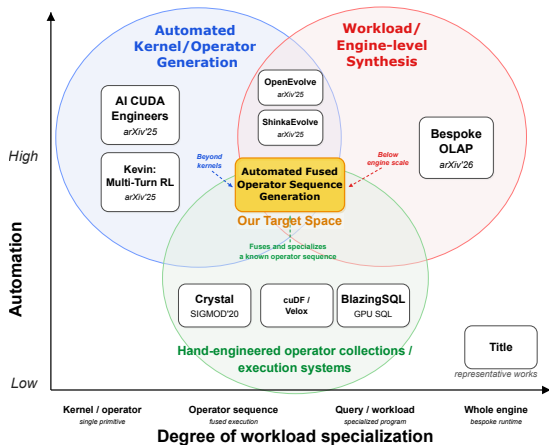
- **Launch** kernel for operator 1;
- **Materialize** intermediate columns;
- Launch kernel for operator 2;
- Read intermediates back from global memory.

Fused pipeline execution

- **Launch one kernel** for a bounded window;
- Keep transient values in **registers or shared memory**;
- Emit only the window output;
- Fall back when fusion is not available.

Fusion reclaims launch overhead and global-memory round trips.

The Design Space



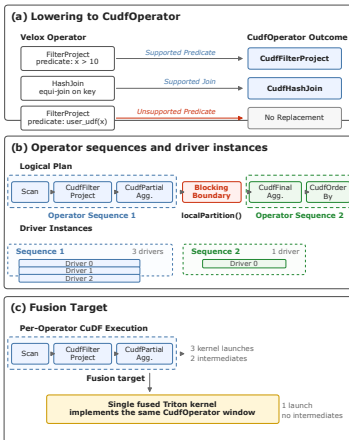
Why Not Just Hand-Write the Kernels?

- Fusion opportunities depend on **query plans, operator windows, types, and layout contracts**;
- Plans change; a hand-written kernel per recurring window does not scale operationally;
- Whole-workload synthesis can be fast, but the generated artifact becomes large and harder to validate;
- We want **small generated artifacts** with a calibrated reference path;

There are inherent tradeoffs between **specificity, performance, and optimization cost**.

System

Execution Target: Velox Pipelines



Three-Stage Workflow

1. **Plan lowering:** extract pipeline windows and emit structural dispatch rules plus C++ fused-operator scaffolds;
2. **Seed generation:** ask a code model to fill each scaffold with glue code and Triton kernel implementations;
3. **Evolutionary refinement:** mutate only the Triton kernel body under the same correctness gates.

Each stage produces an artifact that the next stage can **test mechanically**.

Dispatch Rules Preserve the Existing Engine

- Rules match **structural plan properties**: operator sequence, interfaces, and supported expressions;
- A matched window is replaced by a **fused GPU operator**;
- Failed, unsupported, or unmatched regions execute through the **existing cuDF backend**;
- The core runtime is **kernel-agnostic**; the generated code lives at the edge.

matched window → fused Triton operator otherwise →
cuDF fallback

Example: A Fusion Dispatch Rule

```
{
  "fusion_rules": [{
    "name": "Q1Pipeline1",
    "enabled": true,
    "acceptsGpuInput": true,
    "producesGpuOutput": true,
    "operators": [
      {
        "type": "Aggregation",
        "predicates": [
          "step:eq:final",
          "groupingKeys.count:eq:2",
          "aggregates.count:min:1"
        ]
      },
      { "type": "OrderBy", "predicates": [] }
    ]
  }]
}
```

The JSON rule identifies the **contiguous operator window**; the generated scaffold and equivalence gate enforce the remaining semantic contract.

Before benchmarking

- Patch applies in scope;
- C++ and Python build;
- Fused output matches **unfused Velox/cuDF**.

Stress cases

- Null-heavy inputs;
- Extreme skew;
- Decimal and overflow-sensitive values;
- Empty and tiny inputs;
- Large grouped aggregations.

Only **equivalent candidates** contribute to speedup results.

What the Model Has to Get Right

```
FilterProject -> PartialAggregation -> OrderBy -> Limit  
becomes  
FusedOperator(rule, schema, buffers, triton_kernel)
```

- Implement the **Triton kernel**;
- Respect **Velox-side operator interfaces**;
- Maintain **cuDF-compatible memory and column layout** assumptions;
- Repair failures from **compiler and equivalence feedback**.

This is an **integration task**, not pure code completion.

Evaluation

Experimental Setup

- Benchmark: **TPC-H-derived pipeline windows**;
- Coverage: **90 per-pipeline rule files** across 22 TPC-H queries;
- Comparison set: 21 queries; Q1 is excluded because it is used as a worked prompt example;
- Baseline: **per-operator Velox/cuDF** execution in the same stack;
- Scale factors: SF100, SF300, SF1000.

The comparison isolates **automated fusion inside a fixed execution engine.**

Models Evaluated

Model	Role in the study
Claude-Opus-4.7	strongest generator
GPT-5.3-Codex	frontier coding baseline
Claude-Haiku-4.5	lower-cost proprietary model
Qwen3-30B-A3B-Thinking-2507	open-source coding model

- Each model gets **build and equivalence feedback**;
- Runs stop when a **correct seed** is found or the iteration budget (25) is exhausted.

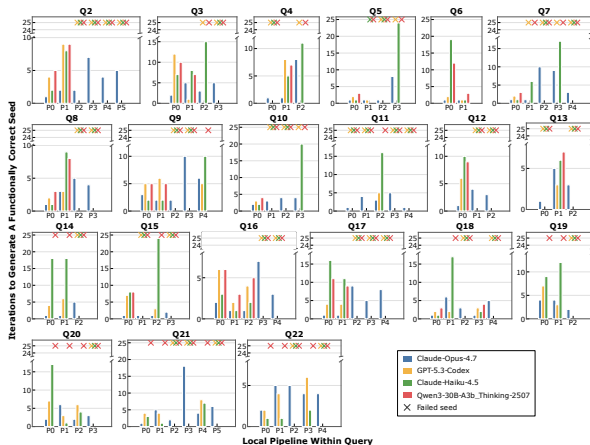
Generation and Optimization Cost

Model	Seed success	Iter./pipe.	\$/iter.	\$/pipe.
Claude-Opus-4.7	100.0%	3.5	\$0.33	\$1.17
GPT-5.3-Codex	47.7%	15.12	\$0.07	\$1.04
Claude-Haiku-4.5	53.4%	15.99	\$0.05	\$0.86
Qwen3-30B-A3B	27.3%	19.9	\$0.01	\$0.22

- Generation cost includes **successful and failed** seed attempts;
- Lower per-token price does not imply lower cost per **usable seed**;
- Refinement cost is tracked separately in Figure 5: search buys additional speedup, but the cost-quality tradeoff is explicit.

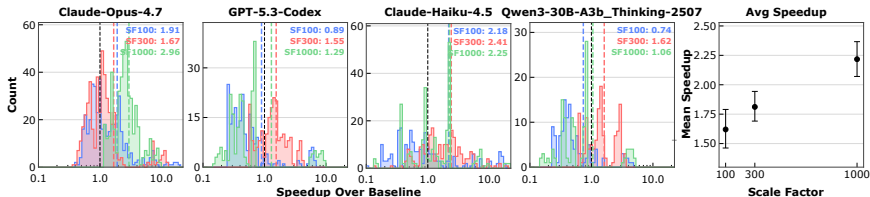
Convergence dominates price: Opus is expensive per call, but cheap per successful pipeline because it needs fewer repair iterations.

Can Models Produce Correct Seeds?

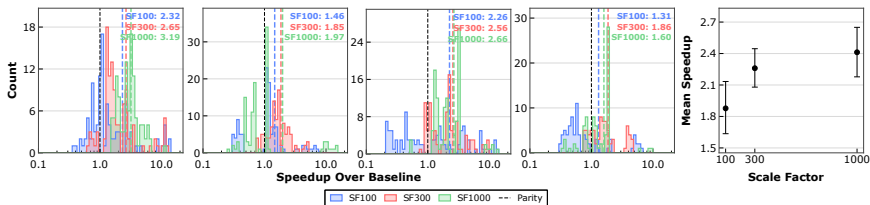


Do Correct Seeds Improve Runtime?

(a) Single Fused Pipeline



(b) Opportunistic Fusion with Fallback



End-to-End Speedups

Configuration	SF100	SF300	SF1000
Claude-Opus, single fused pipeline	1.91×	1.67×	2.96×
Claude-Opus, opportunistic fusion	2.32×	2.65×	3.19×
GPT-5.3-Codex, opportunistic fusion	1.46×	1.85×	1.97×
Claude-Haiku, opportunistic fusion	2.26×	2.56×	2.66×
Qwen3, opportunistic fusion	1.31×	1.86×	1.60×

Correct seeds were optimized first for **equivalence**, yet opportunistic deployment often yields material speedups.

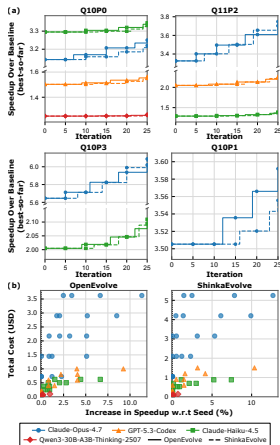
Refinement

Evolutionary Refinement

- Seed kernels are **correct**, but not necessarily optimized;
- **SkyDiscover** routes candidates to OpenEvolve or ShinkaEvolve;
- The search mutates only the **Triton kernel body**;
- The same **preflight, build, equivalence, and benchmark gates** score each candidate;

The refinement loop treats **correctness** as a hard constraint and **speed** as the optimization objective.

Refinement Results



Expert Calibration

Target	Variant	Speedup	% Expert
Q10P3	Per-operator cuDF baseline	1.00×	15.7%
	Claude-Opus-4.7 seed	5.65×	89.0%
	OpenEvolve refined	6.02×	94.8%
	ShinkaEvolve refined	6.09×	95.9%
	Expert hand-tuned	6.35×	100%
Q10P0	Per-operator cuDF baseline	1.00×	26.7%
	Claude-Opus-4.7 seed	3.14×	84.0%
	OpenEvolve refined	3.35×	89.6%
	ShinkaEvolve refined	3.33×	89.0%
	Expert hand-tuned	3.74×	100%

Outlook

Future Directions

- Validate **structural dispatch rules** beyond TPC-H;
- Explore a **single agentic loop** that proposes, validates, and improves fused kernels jointly;
- Combine **physical fusion** with logical plan rewriting.

Generated fused kernels can be introduced into an existing GPU SQL backend **opportunistically**: match a bounded pipeline window, validate against the reference path, deploy when beneficial, and fall back otherwise.

This keeps the engineering advantages of Velox/cuDF while reclaiming performance lost at operator boundaries.

Questions?

Corey Lammie
`corey.lammie@ibm.com`