Accelerating Python for real-time ML with Chalk

Chase Haddleton, Engineer, Chalk

FEATURE STORE SUMMIT 2025

Organized by HOPSWORKS

right data
right place
right time

FEATURE STORE SUMMIT 2025

Organized by HOPSWORKS

why is real-time ML important?

Fraud detection, RecSys, portfolio risk (etc.) solutions need both

- Fresh features
- Low-latency < 10ms

Out of date or slow == real cost to the business

> Inevitably forced to build a bespoke (custom) solution

why is real-time ML hard?

Need deep expertise across very different domains

- ML, low-latency systems, scale-out architectures

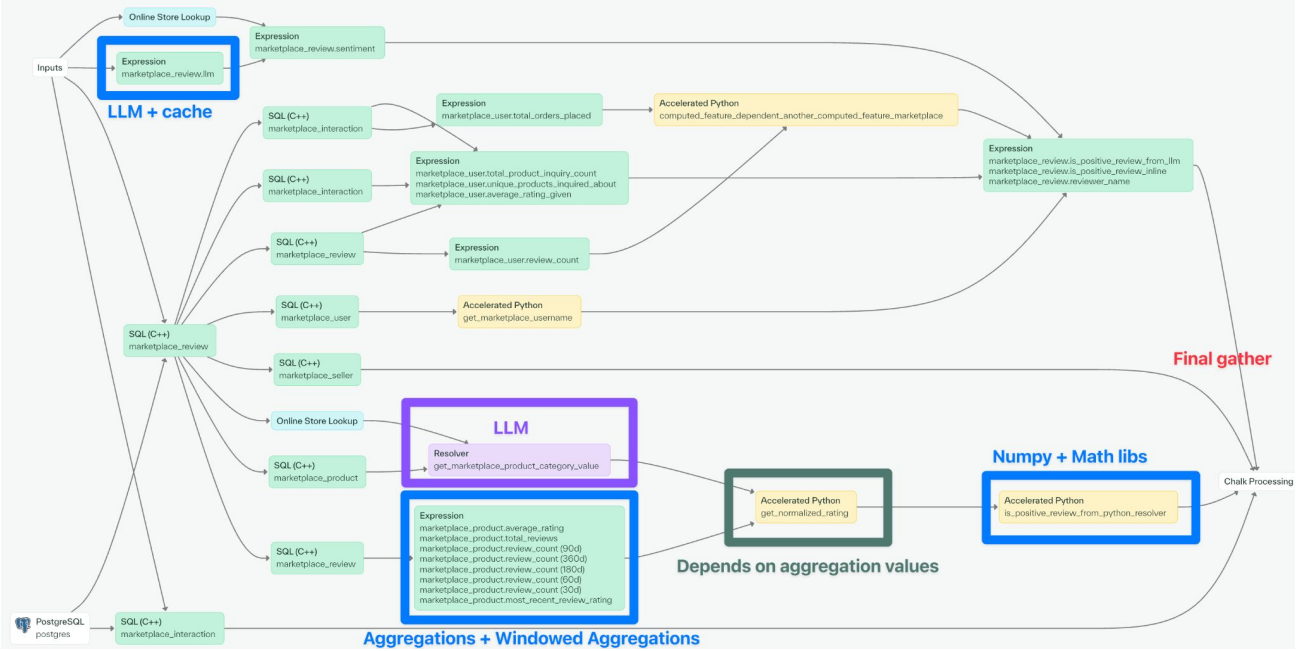
Sprawling tech, stitched into brittle pipelines

- Pytorch, SciKit, Sagemaker, Keras, Data wrangling
- Scala, Rust, dbt, Spark, Airflow, Iceberg
- Kubernetes, Terraform, KEDA, Workflow orchestration

In addition, ML teams want to

- Ship models quickly
- Incorporate new data sources and types (unstructured)
- React to real-time signals

chalk query --in review.id=241 --out normalized_rating –-out <other_features>



Features

features needs to be easy for the user to express i.e. not YAML

```
@features
class Transaction:
    id: int

    # user_id: int
    user_id: User.id
    user: User

    total: float
    is_expensive: bool = _.total > 1000

    created_at: datetime
```

```
@features
class User:
    id: id
    name: str

    # inferred with Transaction.user_id
    transactions: DataFrame[Transaction]

    total_spend: float = _.transactions[_.total].sum()
    most_recent_transaction: float = F.max_by(
        _.transactions[_.total],
        sort=_.created_at,
    )
```

Resolvers

Chalk expressions

Python

SQL

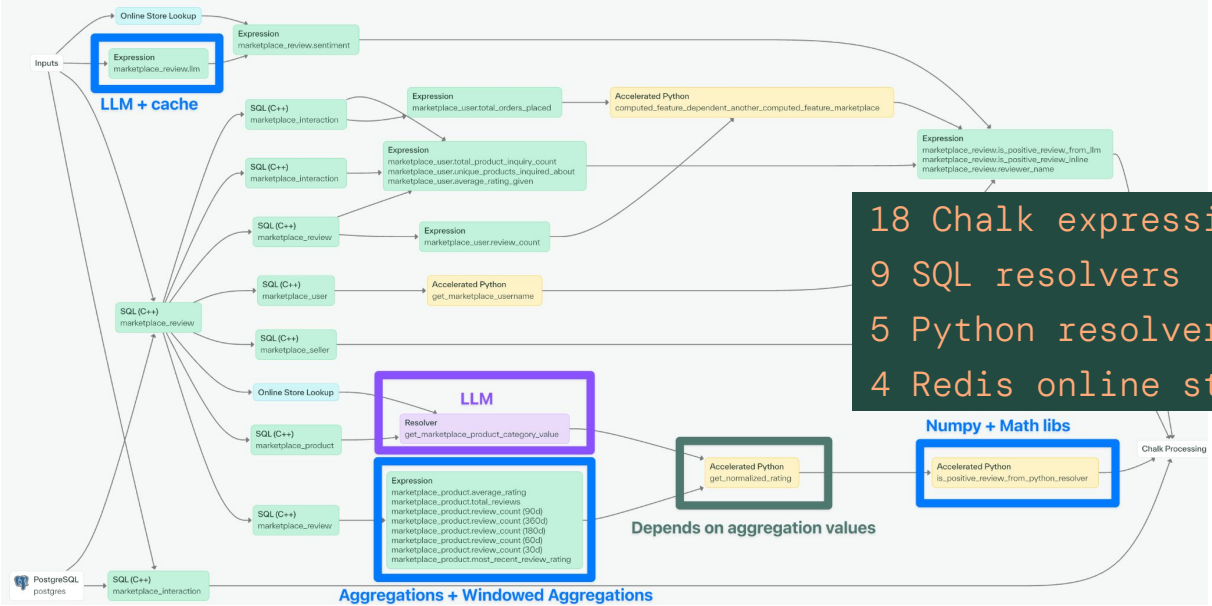
```python
def compute_total(
  subtotal:   Transaction.subtotal,              # float
  discount:   Transaction.discount,              # float | None
  tax_rate:   Transaction.locale.sales_tax_rate, # float
  is_taxable: Transaction.taxable,               # bool
) → Transaction.total:

  if discount is not None:
    # Apply any discount, but don't allow it to become negative.
    subtotal = max(subtotal - discount, 0.0)

  if not is_taxable or tax_rate is None:
    return subtotal

  return subtotal * (1.0 + tax_rate)

```

```python
@features
class User:
    id: id
    name: str

    # inferred with Transaction.user_id
    transactions: DataFrame[Transaction]

    total_spend: float = _.transactions[_.total].sum()
    most_recent_transaction: float = F.max_by(
        _.transactions[_.total],
        sort=_.created_at,
    )
```

enrich_transaction.chalk.sql

```sql
-- resolves: Transaction
-- source: bigquery
select
    txn_id as id,
    desc as description
    receipt_id,
from enriched_transaction_memos
```

chalk query --in review.id=241 --out normalized_rating –-out <other_features>




```
18 Chalk expressions (UDFs)
9 SQL resolvers
5 Python resolvers (4 UDFs)
4 Redis online store lookups
```

Symbolic Python Interpreter

many Python resolvers are relatively straightforward

```python
def compute_total(
  subtotal:   Transaction.subtotal,              # float
  discount:   Transaction.discount,              # float | None
  tax_rate:   Transaction.locale.sales_tax_rate, # float
  is_taxable: Transaction.taxable,               # bool
) -> Transaction.total:

  if discount is not None:
    # Apply any discount, but don't allow it to become negative.
    subtotal = max(subtotal - discount, 0.0)

  if not is_taxable or tax_rate is None:
    return subtotal

  return subtotal * (1.0 + tax_rate)

```

Symbolic Python Interpreter

Converts Scalar Python Resolver → Expression

- Resolvers already have a strict arrow schema, as do expressions
- **Convert at plan time**
- Avoids Python completely at query time

Python Resolver → Expression (example)

```python
if discount is not None:
  subtotal = max(subtotal - discount, 0.0)
if not is_taxable or tax_rate is None:
  return subtotal
return subtotal * (1.0 + tax_rate)
```

```
if(
 or(
   not(col("I")),
   is_null(col("T")),
 ),
 if(
   not(is_null(col("D"))),
   max(col("S") - col("D"), lit(0.0)),
   col("S"),
 ),
 if(
   not(is_null(col("D"))),
   max(col("S") - col("D"), lit(0.0)),
   col("S"),
 ) * (1.0 + col("T")),
)
```

Rewrites de-duplicate common sub-expressions

Python Resolver → Expression (example)

SymbolicValue:

- Velox Expression + Python Type
- Initially col(...) expressions
- Immutable, side-effect free
  - Always refer to original value

Environment

```
discount    = col("D")/float
subtotal    = col("S")/float
is_taxable = col("I")/float
tax_rate    = col("T")/float?
```

Environment

```
discount    = abc(...)
subtotal    = max(col("S") - 10.0, 0.0)
is_taxable = ghi(...)
tax_rate    = jkl(...)
```

Symbolic Python Interpreter

Fallback to running Python with process pools

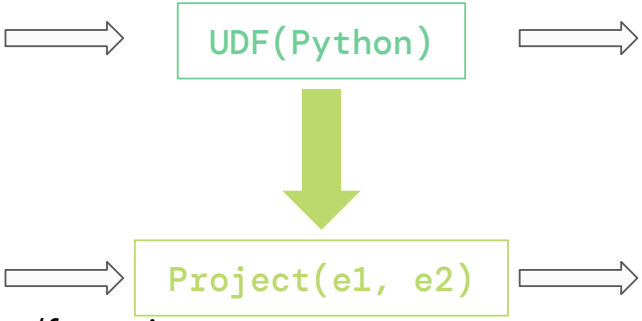
- For unsupported functions e.g. openai.responses.create
- Functions with runtime-validated preconditions

“Gas Limit” estimates total amount of work (at query time + runtime) to determine whether conversion is a good idea

Symbolic Python Interpreter

Python as a DSL for writing expressions!

- Can run as regular Python functions outside of Chalk
- Lift Python semantics verbatim into Velox

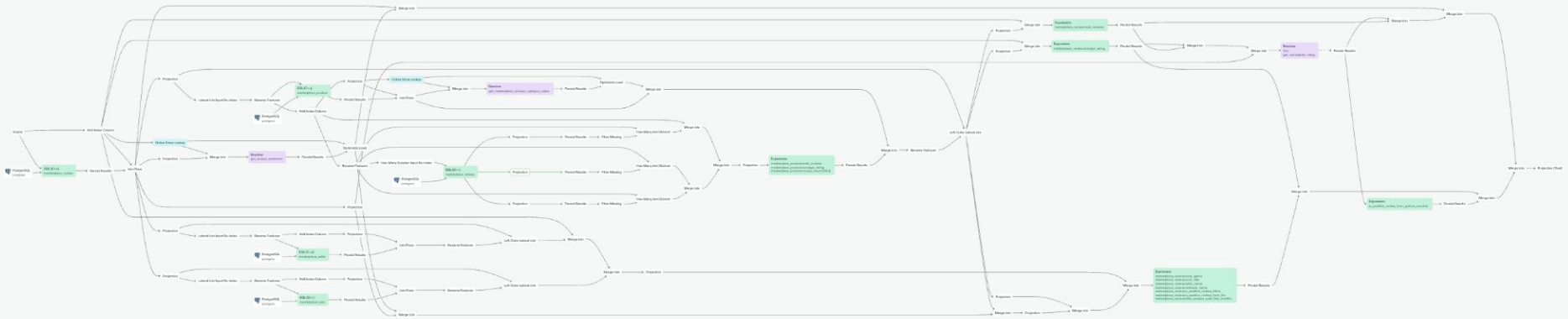


Benefits

- No GIL
- No heavy per-object heap representation
- Falls back if it encounters any unhandled expression/function
- Becomes SIMD!

Logical Plan

10+ Python Resolvers, 5 Postgres lookups, 2 Redis Online Store lookups, 68ms* response time



* Latency as of march 2025

Velox Plan