Transpiling PL/pgSQL UDF in DuckDB

Adrian Abedon, Yuchen Liu, Arvin Wu

Goal: We want to generate C++ code for PL/pgSQL UDFs

```sql
CREATE OR REPLACE FUNCTION add_one(i integer)
RETURNS integer AS
$$
BEGIN
    RETURN i + 1;
END;
$$
LANGUAGE plpgsql;
```

```cpp
int64_t add_one(int64_t a) {
    return a + 1;
}
```

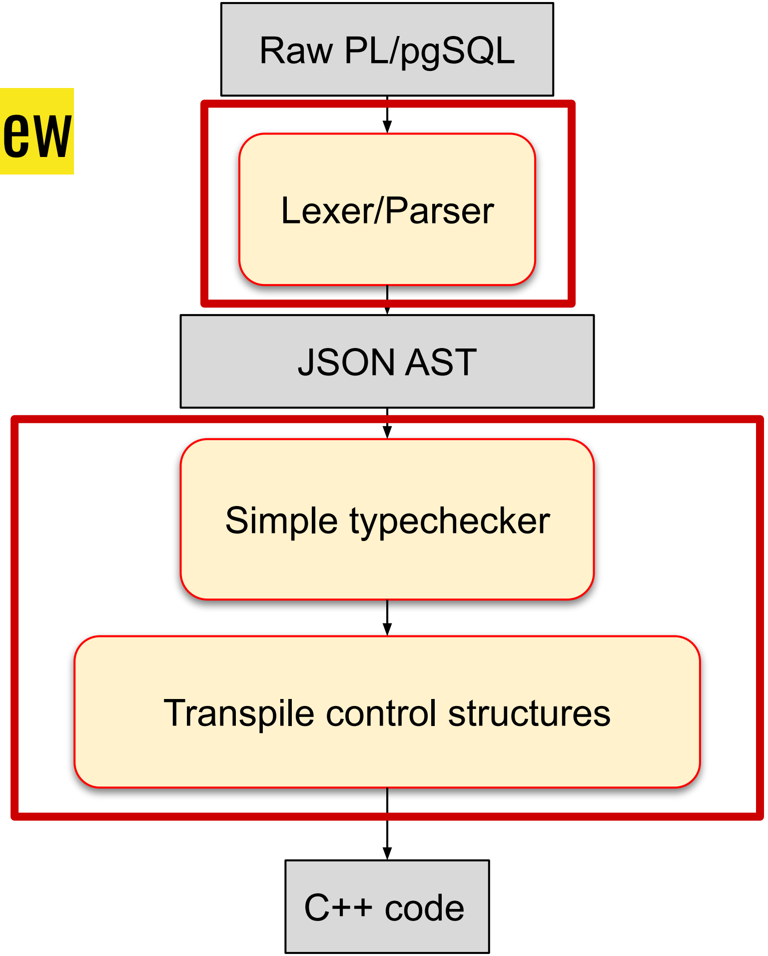
Not supported by DuckDB

Supported by DuckDB

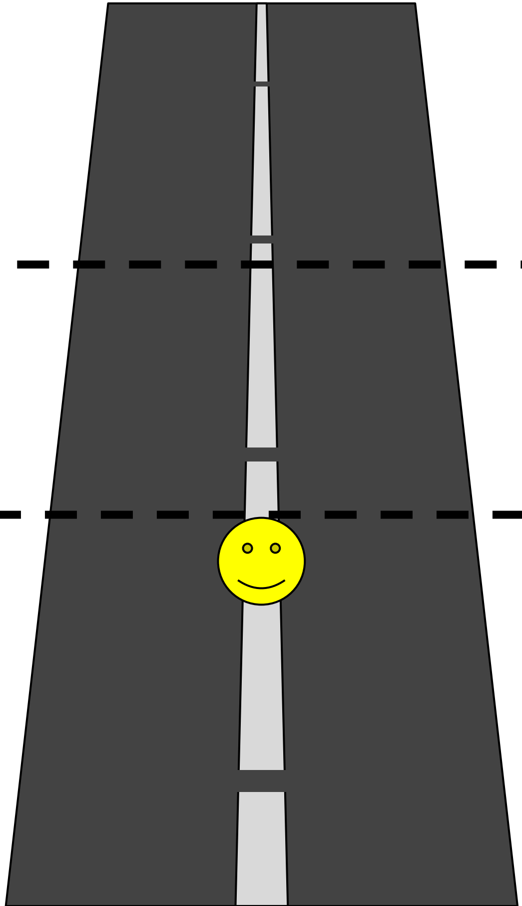
Further Goals

- Support DuckDB vectorization
- Inline DuckDB operators during compilation

Current Architecture Overview



Project Roadmap



Phase I

Transpile PL/pgSQL into a scalar UDF

- Essentially the control flow in the UDF
- Pass SQL queries not parsed to be handled by DuckDB

Phase II

Transpile PL/pgSQL into a vectorized UDF

- Still calling DuckDB to handle queries

Phase III

Compile SQL queries through operator inlining

- Gradually support the compilation of certain queries
  - Comparators >, <
  - Numeric Operators +, -, *
  - DuckDB built-in functions

Phase I: Transpile PL/pgSQL into a scalar UDF

- **Focus was on transpiling control structures**
  - Function body and arguments
  - If statements
  - For/While loops
- **Let DuckDB handle queries**
  - ```
    return a+10;
    ```
    →
    ```
    return con.Query("select $1+10", a);
    ```
- **Issues with DuckDB scalar UDF API**
  - Only supports 3 arguments
  - Loose precision with DECIMAL type
  - Difficult to interpret certain types such as DATE since they are passed as int32
    ```

Phase II: Transpile PL/pgSQL into a vectorized UDF
Motivation

- All the drawbacks of scalar UDF
- The bottleneck of performance: **HUGE** overhead of calling DuckDB's query
  - ~1 ms ≈ 10^6 CPU cycles for each simple query: select l_quantity+1

By switching to the vectorized UDF framework, we can share that overhead in the vector

- Most DuckDB operators are vectorized
  - ```
    void Equals(Vector &left, Vector &right, Vector &result, idx_t count);
    ```
  - Important when operators are inlined in Phase III
    ```

Phase II: Transpile PL/pgSQL into a vectorized UDF

- **Rewrite control structures to support vectorization**
  - Function body and arguments
  - If statements
  - For/While loops
- **Use vectorized prepared statements**
- **Explore how to write vectorized UDF**
  - Interact directly with duckdb vectors
    - Flat vector
    - Constant vector
    - Dictionary vector

Phase II: Transpile PL/pgSQL into a vectorized UDF

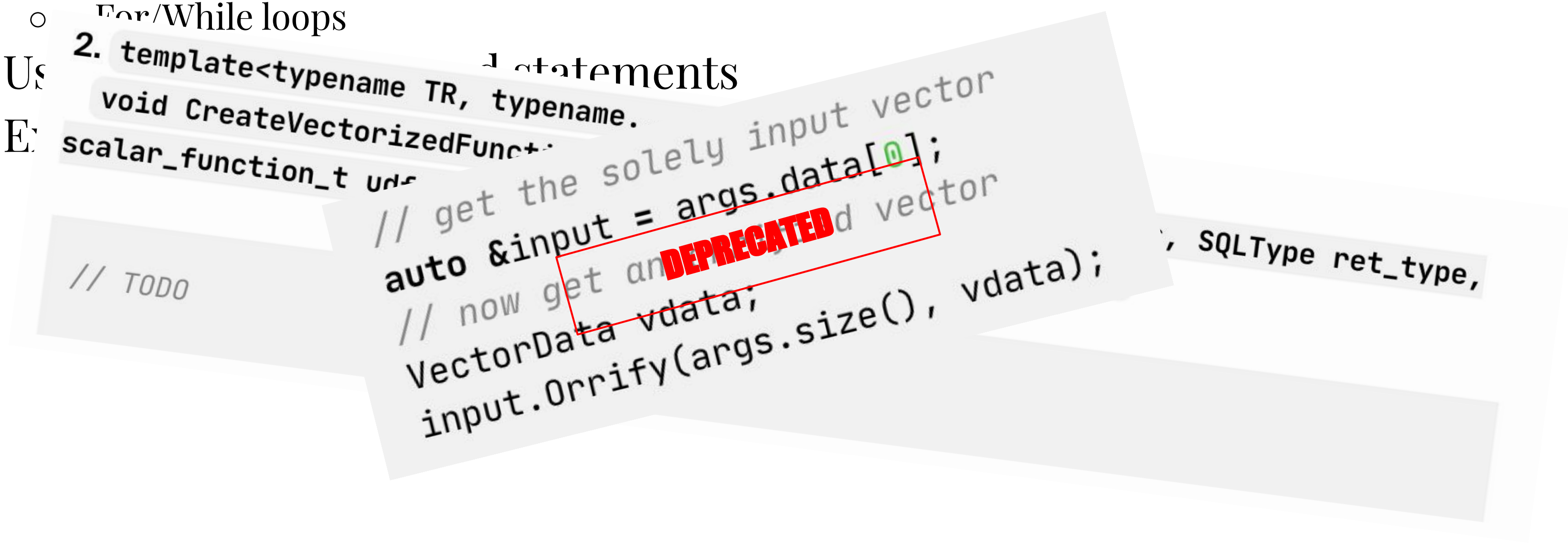
- Have to rewrite control structures to support vectorization
  - Function body and arguments
  - If statements
  - For/While loops
- Us… …d statements
- E…

2.
```
template<typename TR, typename... Args>
void CreateVectorizedFunction(string name, vector<SQLType> args, SQLType ret_type,
scalar_function_t udf_func, SQLType varargs = SQLType::INVALID)

// TODO
```

Phase II: Transpile PL/pgSQL into a vectorized UDF

- Have to rewrite control structures to support vectorization
  - Function body and arguments
  - If statements
  - For/While loops
- Us… …d statements
- E…



Vectorized control structures

- Vectorizing arbitrary control structures is difficult
- Our approach was to keep track of "**active lanes**" (std::vector<bool>)
  - Set upon if condition, return, break, or continue
- If a lane is inactive, no computation happens on that lane

Example:

```
a = [5,10,4,13]
b = [1,11,3,14]
if(a < b):                  -> active lanes = [0,1,0,1]
    a = a+b                 -> a = [5,21,4,27]
```

```
create function line_count(oprio char, mode varchar)
returns int as $$
declare val int = 0;
begin
    if(mode = 'high') then
        if(oprio = '1-URGENT' OR oprio = '2-HIGH') then
            val = 1;
        end if;
    elsif(mode = 'low') then
        if(oprio <> '1-URGENT' AND oprio <> '2-HIGH') then
            val = 1;
        end if;
    end if;
    return val;
end $$
LANGUAGE PLPGSQL;
```

```
std::vector<duckdb::Value> tempvar116 = query26(shipdate, &active2, &returns1, NULL, NULL);
std::vector<bool> active3 = active2;
for (size_t tempvar115 = 0; tempvar115 < tempvar116.size(); tempvar115++)
{
  // if we are in a while loop, we don't need to check loop_active/continues since
  // the tempvar116 will be appropriately set to null.
  if (!tempvar116[tempvar115].IsNull())
  {
    active3[tempvar115] = active3[tempvar115] && (tempvar116[tempvar115].template GetValue<bool>());
  }
}

/** RETURN **/
std::vector<duckdb::Value> tempvar114 = const_vector_gen(0);
for (size_t tempvar113 = 0; tempvar113 < args.size(); tempvar113++)
{
  if (active3[tempvar113] && !returns1[tempvar113])
  {
    // note that this lane has returned
    returns1[tempvar113] = true;
    // set its return value
    return_values[tempvar113] = tempvar114[tempvar113];
  }
}
for (size_t tempvar115 = 0; tempvar115 < active3.size(); tempvar115++)
{
  if (!tempvar116[tempvar115].IsNull())
  {
    active3[tempvar115] = !active3[tempvar115];
```

Phase III: Compile queries

- **Not yet implemented**
- Support only queries that do not access a table
  - i.e. `select (x*2.0) < 20;`
- Inline DuckDB operators and functions
  - /, *, <, +, =,...
  - date_add, date_part, cast
- Current vectorized transpiler provides a great foundation to add compilation

Correctness Test

- Run TPC-H queries with and without UDF's and compared results
  - Queries 1, 3, 4, 5, 6, 7, 9, 10, 12, 14, 19 from [Froid Preprint]

Query 3 with no UDF's

```
select
    l_orderkey,
    sum(l_extendedprice * (1 - l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_mktsegment = 'BUILDING'
    and c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and o_orderdate < date '1995-03-15'
    and l_shipdate > date '1995-03-15'
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
limit 10;
```

Query 3 with UDF's

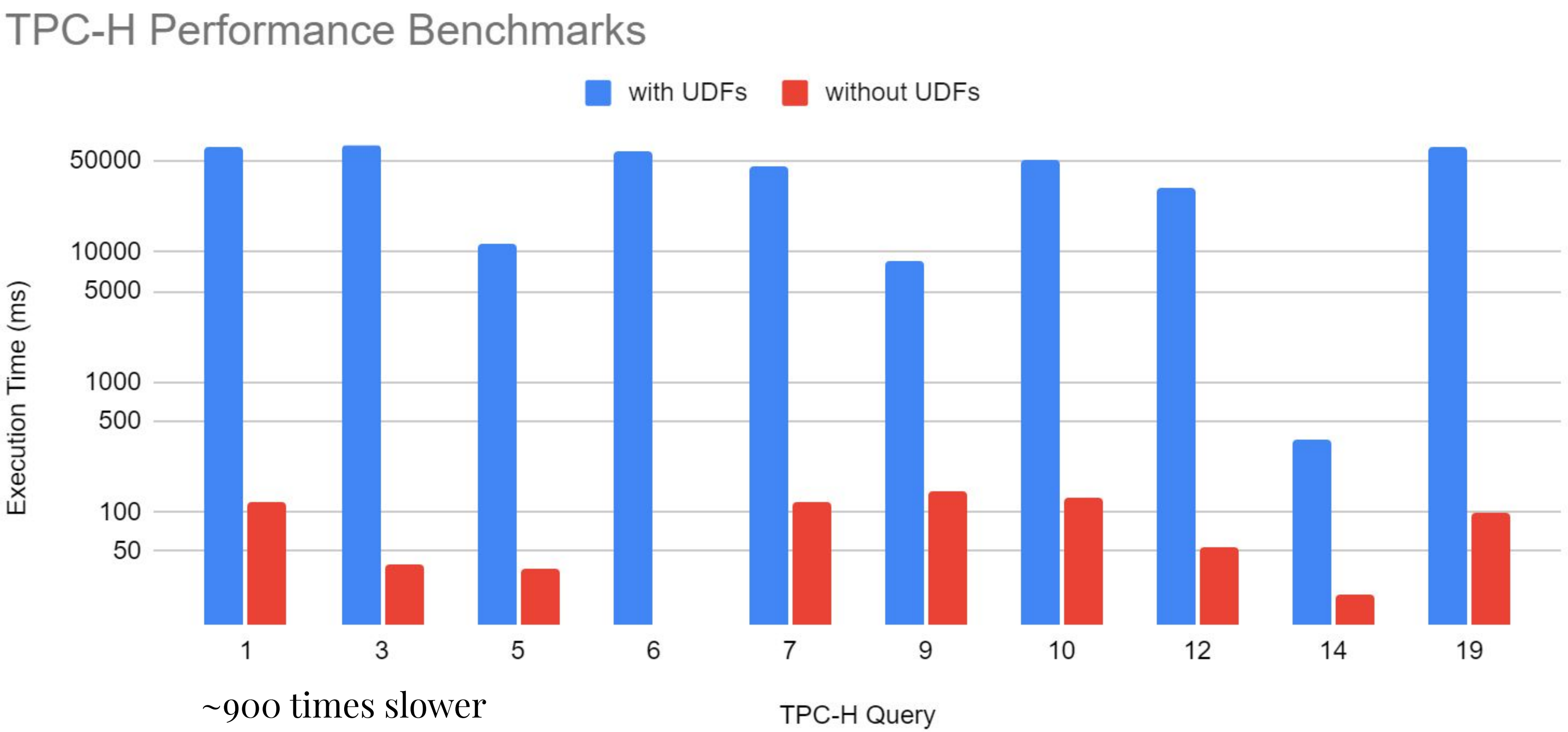
```
select
    l_orderkey,
    sum(discount_price(l_extendedprice, l_discount)) as revenue,
    o_orderdate,
    o_shippriority
from
    customer,
    orders,
    lineitem
where
    c_custkey = o_custkey
    and l_orderkey = o_orderkey
    and q3conditions(c_mktsegment, o_orderdate, l_shipdate) = 1
group by
    l_orderkey,
    o_orderdate,
    o_shippriority
order by
    revenue desc,
    o_orderdate
limit
    10
```

Performance without Query Compilation

- Current version is significantly bottlenecked by making calls to DuckDB inside of UDF's

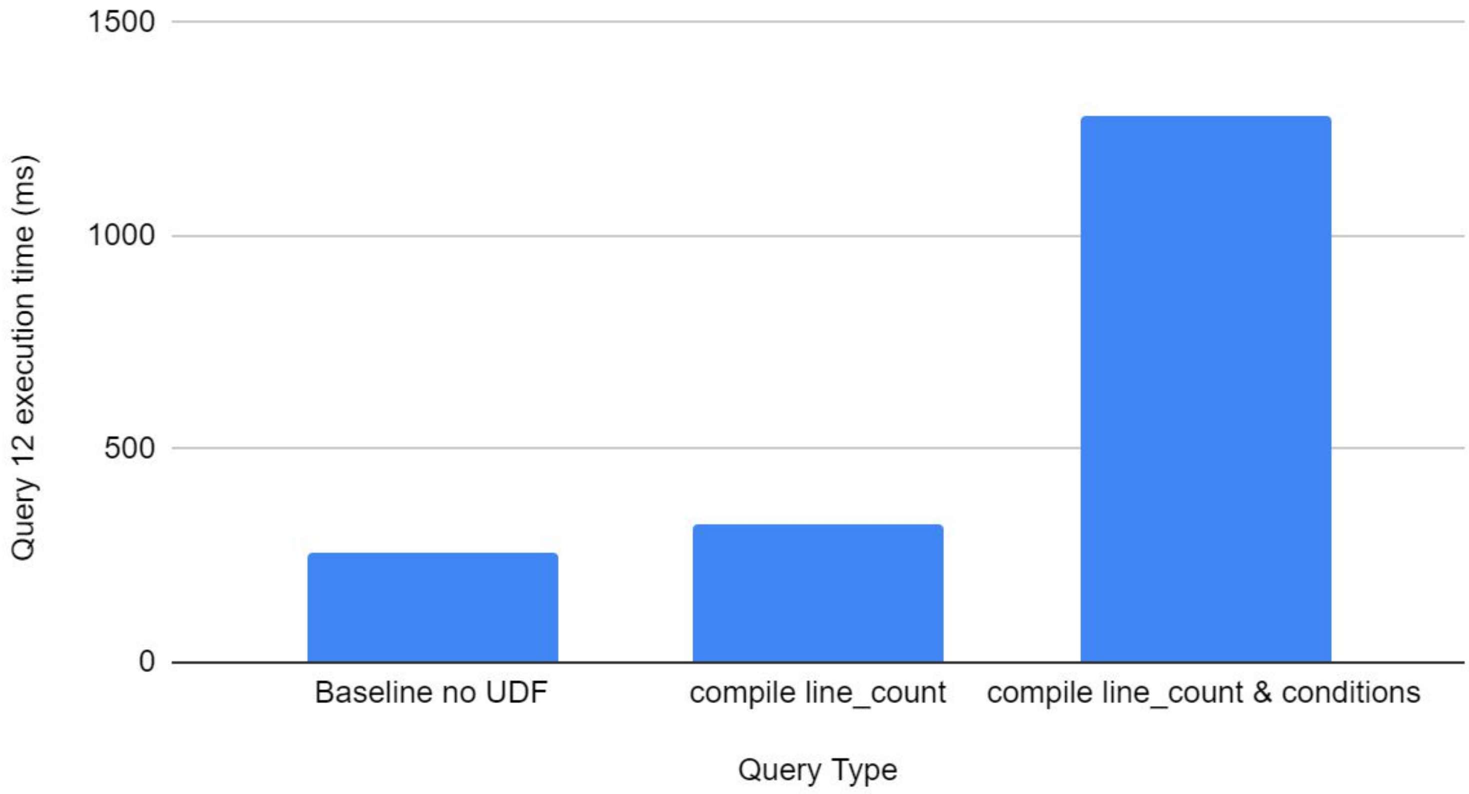
TLDR: Its slow right now but it can be pretty fast

Performance without Query Compilation

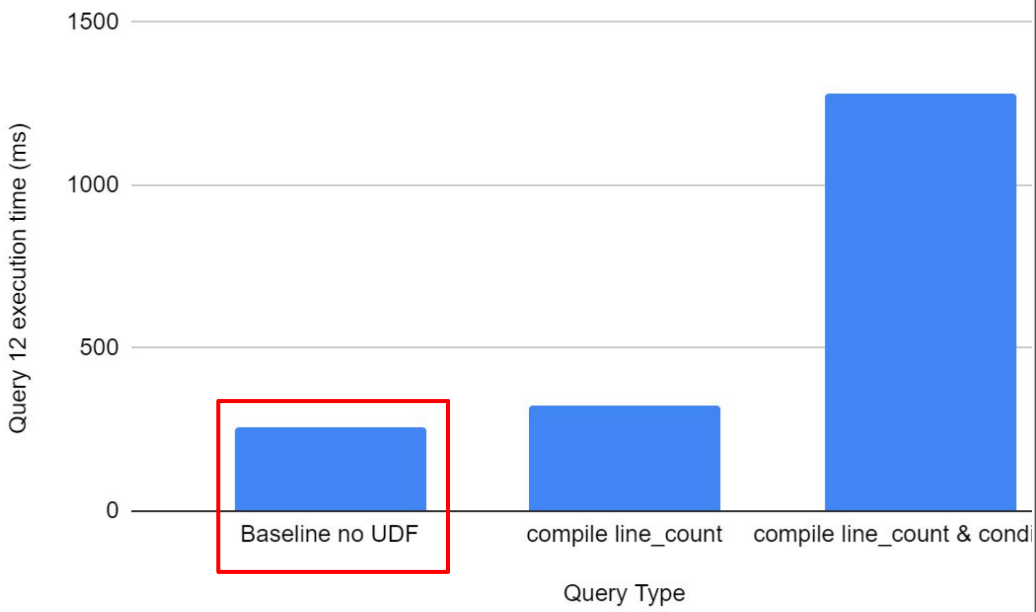


~900 times slower

Query 12 Performance with full UDF Compilation

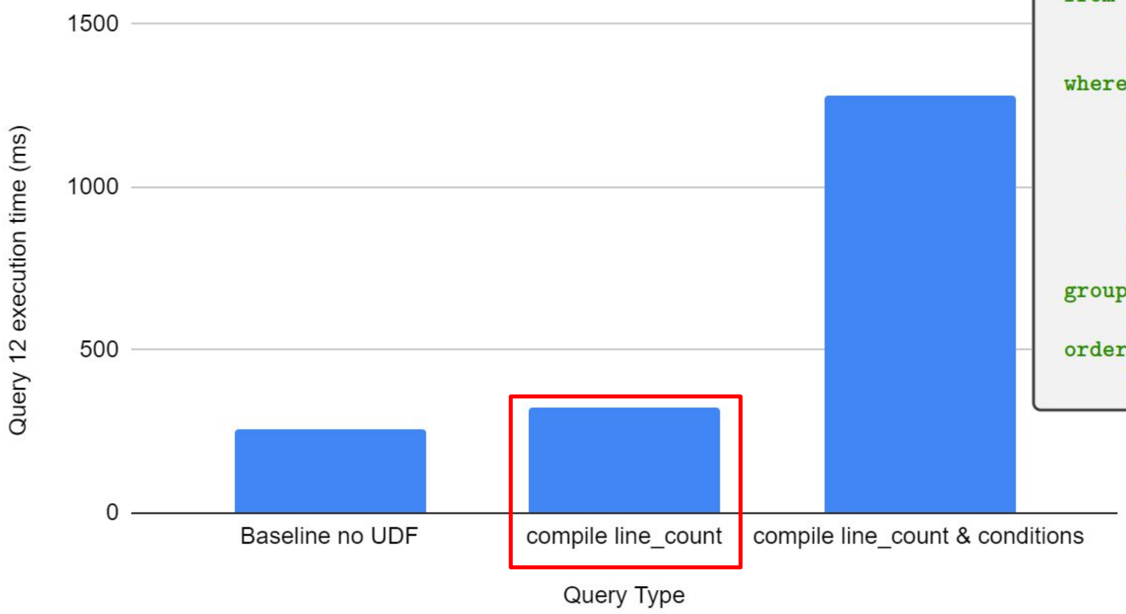


Query 12 Performance with full UDF Compilation



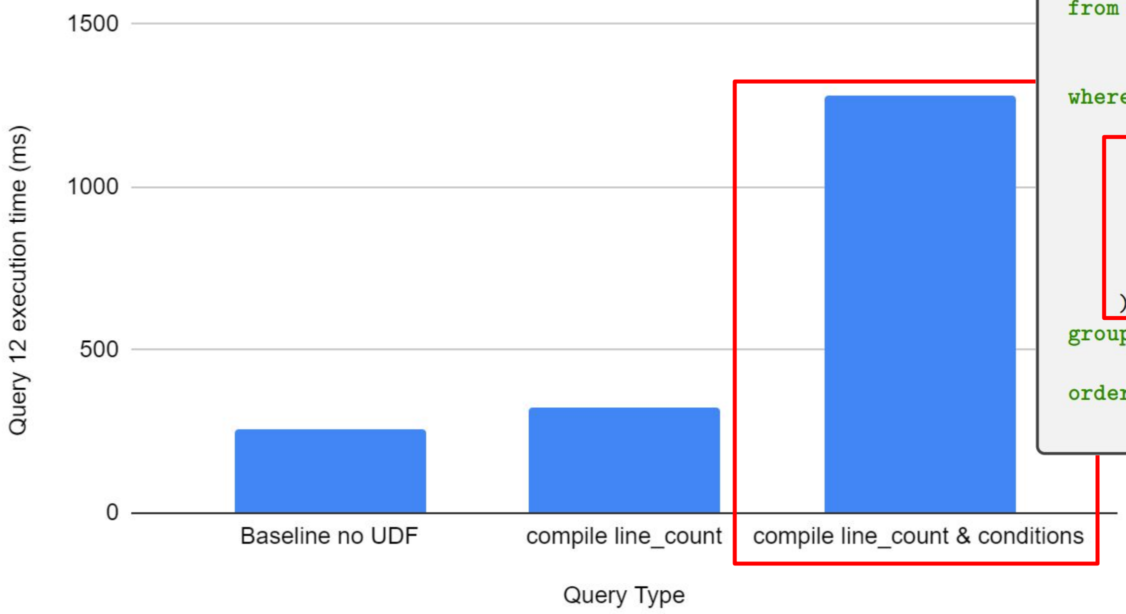

```sql
select
      l_shipmode,
      sum(case
            when o_orderpriority = '1-URGENT'
                  or o_orderpriority = '2-HIGH'
                  then 1
            else 0
      end) as high_line_count,
      sum(case
            when o_orderpriority <> '1-URGENT'
                  and o_orderpriority <> '2-HIGH'
                  then 1
            else 0
      end) as low_line_count
from
      orders,
      lineitem
where
      o_orderkey = l_orderkey
      and l_shipmode in ('MAIL', 'SHIP')
      and l_commitdate < l_receiptdate
      and l_shipdate < l_commitdate
      and l_receiptdate >= date '1994-01-01'
      and l_receiptdate < date '1994-01-01' + interval '1' year
group by
      l_shipmode
order by
      l_shipmode;
```

Query 12 Performance with full UDF Compilation



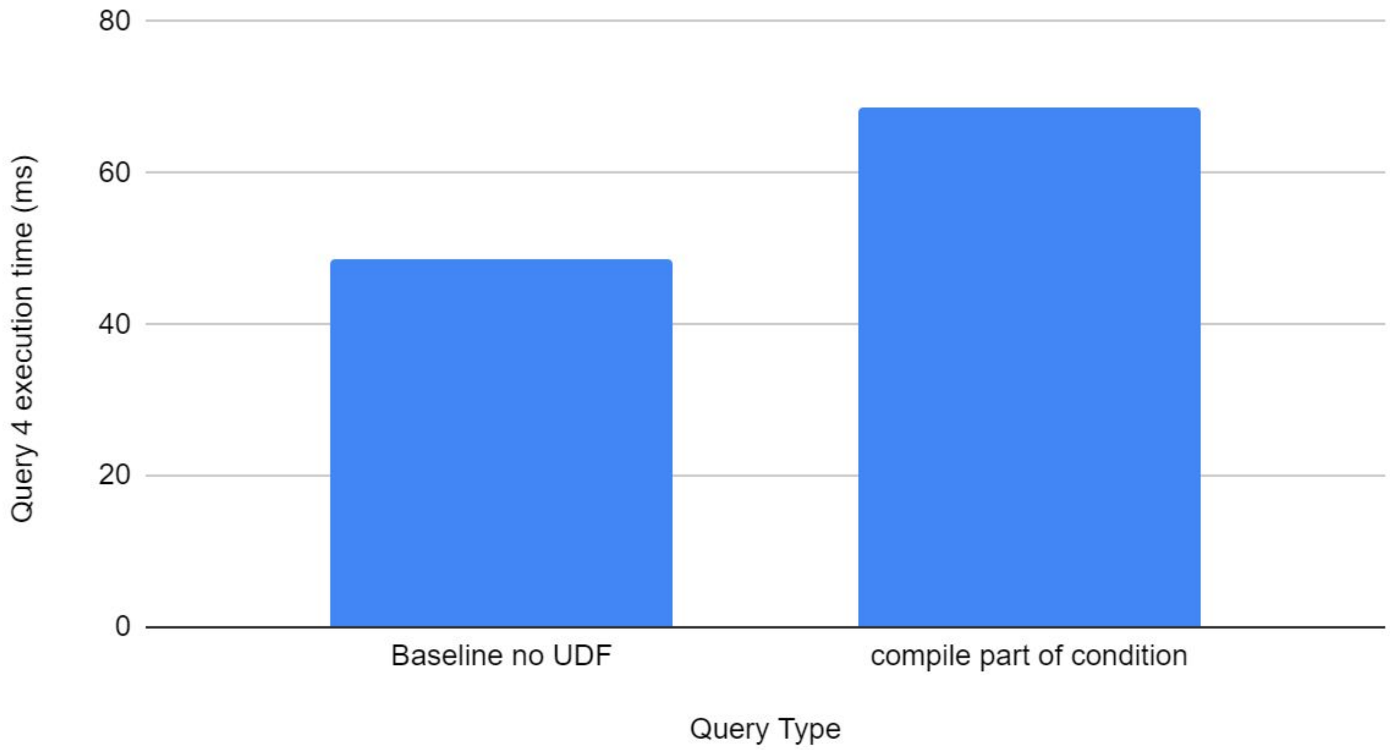

```sql
select
      l_shipmode,
      sum(line_count(o_orderpriority, 'high')) as high_line_count
      sum(line_count(o_orderpriority, 'low')) as low_line_count
from
      orders,
      lineitem
where
      o_orderkey = l_orderkey
      and l_shipmode in ('MAIL', 'SHIP')
      and l_commitdate < l_receiptdate
      and l_shipdate < l_commitdate
      and l_receiptdate >= date '1994-01-01'
      and l_receiptdate < date '1994-01-01' + interval '1' year
group by
      l_shipmode
order by
      l_shipmode;
```

Query 12 Performance with full UDF Compilation




```sql
select
    l_shipmode,
    sum(line_count(o_orderpriority, 'high')) as high_line_count
    sum(line_count(o_orderpriority, 'low')) as low_line_count
from
    orders,
    lineitem
where
    o_orderkey = l_orderkey
    and q12conditions(
        l_shipmode,
        l_commitdate,
        l_receiptdate,
        l_shipdate
    ) = 1
group by
    l_shipmode
order by
    l_shipmode;
```

Query 4 Performance with full UDF Compilation



Conclusions from Benchmarks

- Calling DuckDB to execute queries in UDF’s is way to slow
- Need to compile DuckDB operators
- Compiling UDFs makes sense if it is performing general computation, such as **inside the SELECT**
- Compiling UDFs does NOT make sense if it is used to **filter rows**
    - DuckDB (and other databases) will not be able to apply
        - block skipping
        - short circuiting predicate evaluations with AND
    - Better to use Froid/APFEL approach
- We can potentially improve performance with UDFs

Code Quality

- The framework we have developed for transpilation is extensible
  - Can choose to compile certain operators
  - For unsupported operators we can call DuckDB
- All C++ building blocks come from yaml templates
  - Can easily modify C++ output
- We feel that code is past the prototype stage but far from production ready
  - Clean up code
  - Generated C++ code is not formatted

Future Work

- Compile DuckDB operators
- Support more PL/pgSQL
  - nested blocks
  - block labels
- Support UDF calls within UDF’s
  - Limitation of current approach of calling DuckDB for queries
  - This will come with compilation

Thank you