Evergreen

Efficient Claim Verification for Semantic Aggregates

Alexander Lee*, Benjamin Han, Shayak Sen, Sam Yeom, Uğur Çetintemel, Anupam Datta

Semantic Query Processing Engines

E.g., Aryn (Anderson et al.), UQE (Dai et al.), BigQuery (Google), ThalamusDB (Jo and Trummer), Palimpzest (Liu et al.), VectraFlow (Lu et al.), LOTUS (Patel et al.), DocETL (Shankar et al.), **Cortex AISQL (Snowflake)**[1], CAESURA (Urban and Binning), …

Query engines with **semantic processing** capabilities

Semantic operators: traditional operators augmented with LLMs

[1] Liskowski et al., arXiv (2025)

Semantic Aggregation

Reduce a relation of tuples with an LLM-based function

```
SELECT AI_AGG(review, 'Summarize the reviews for this
                restaurant') as summary
FROM reviews;
```

```
("Flowing Tide is a popular nautical-themed pub chain…The
restaurant's service is inconsistent. For example, some
customers wait 20+ minutes to be acknowledged…")
```

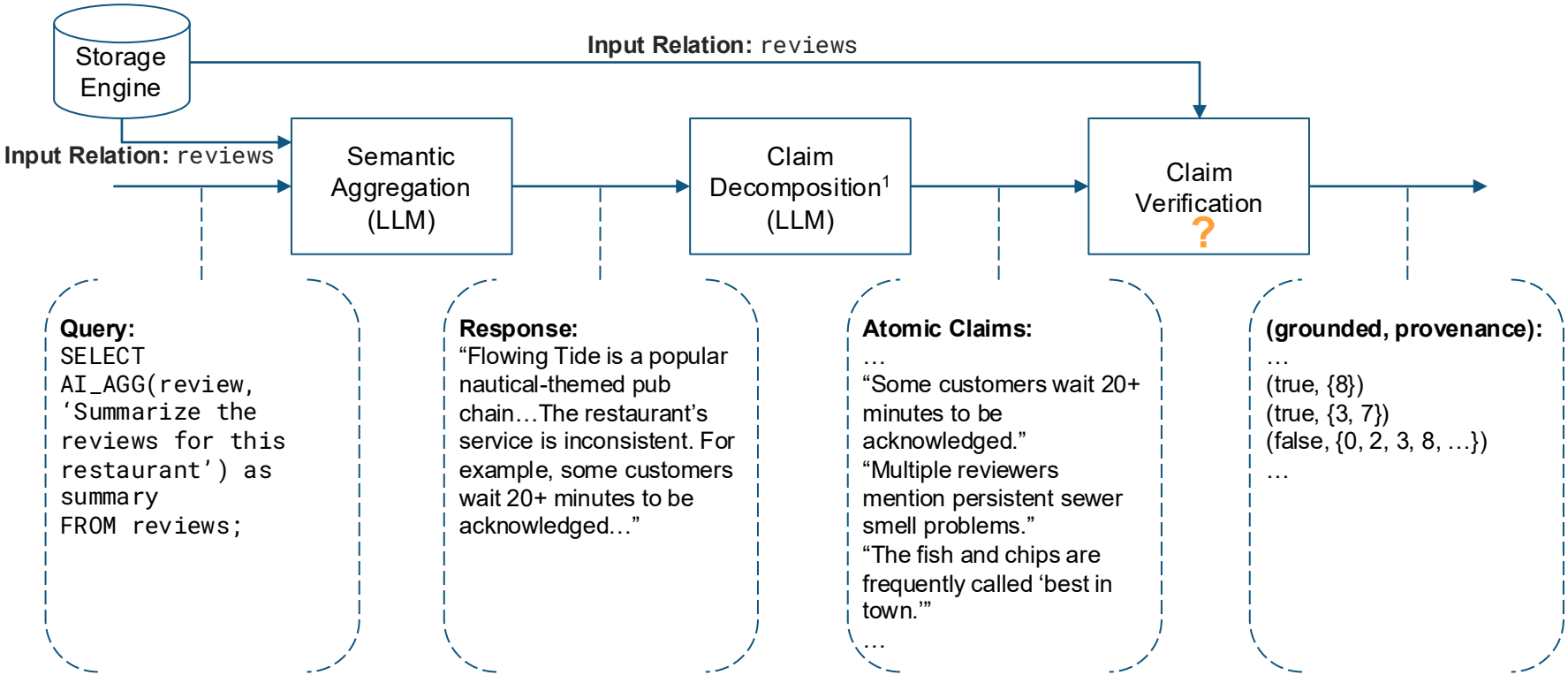
Claim Verification for Semantic Aggregates

- “Some customers wait 20+ minutes to be acknowledged.”
- “Multiple reviewers mention persistent sewer smell problems.”
- “The fish and chips are frequently called ‘best in town.’”
- “Happy hour offers $6 appetizers.”
- …

Some claims may be **ungrounded**

Goal: **verify** each claim is **grounded** in the input relation

Claim Generation & Verification



[1] Min et al., EMNLP (2023)

Approaches to Claim Verification

Off-the-shelf LLM[1]

✗ Faces **context window** limitations

Retrieval-augmented LLM[2]

✗ Lacks **symbolic** processing

Tool-augmented LLM[3]

✗ Bears responsibility of **optimization** and **provenance capture**

Query-augmented LLM[4]

✗ Supports only **traditional** database operators

[1] Min et al., EMNLP (2023); Tang et al., EMNLP (2024) [2] Khattab et al., NeurIPS (2021) [3] Pan et al., ACL (2023); Zhang et al., arXiv (2025) [4] Jayasekara and Trummer, VLDB (2025)

Evergreen

Insight: treat claims as **semantic verification queries**

Advantages:

- Semantic operators for **semantic** processing
- Traditional operators for **symbolic** processing
- Declarative **optimization** for reducing # of LLM calls (by 4.3× or more)
- Reliable **provenance capture** for auditability

Claim Structure

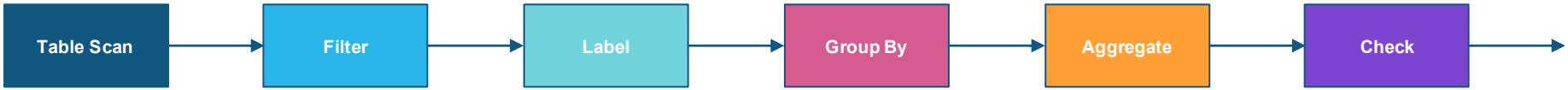
“The fish and chips are frequently called ‘best in town.’”

Components (observed from Snowflake workloads):

- **Group**(s) of tuples
  - E.g., reviews that mention fish and chips
- **Predicate**(s) about a group’s tuples
  - E.g., whether the review calls the fish and chips “best in town”
- **Quantifier**(s) over a group’s tuples that satisfy a predicate
  - E.g., frequently called “best in town”
- **Comparison**(s) of quantities between groups, **nested quantifiers**, …

Claims as Queries

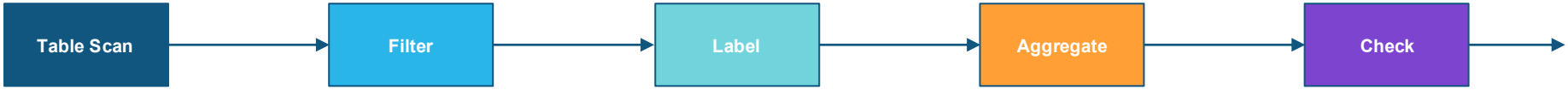
Compile each natural language claim into a **semantic verification query**



1. Filter tuples to define **groups**
2. Label tuples to define **predicates**
3. Group tuples by labels for nested **quantifiers**
4. Aggregate tuples satisfying a predicate to compute **quantities**
   - Functions: `bool_or(predicate)`, `bool_and(predicate)`, `count_if(predicate)`, `proportion(predicate)`
5. Check **quantifiers** or **comparisons**

Verification Query: Example

“The fish and chips are frequently called ‘best in town.’”




```
reviews_df
.filter(prompt(“the {review} mentions the restaurant’s fish and chips”))
.map(prompt(“determine whether the {review} calls the fish and chips ‘best in town’”).alias(“best_in_town”))
.aggregate([proportion(col(“best_in_town”)).alias(“best_in_town_proportion”)])
.check(col(“best_in_town_proportion”) >= 0.1)
```

Optimizations

Early stopping

Relevance sorting

Estimation with confidence intervals (CIs)

Today's talk

Semantic caching

Radius filtering

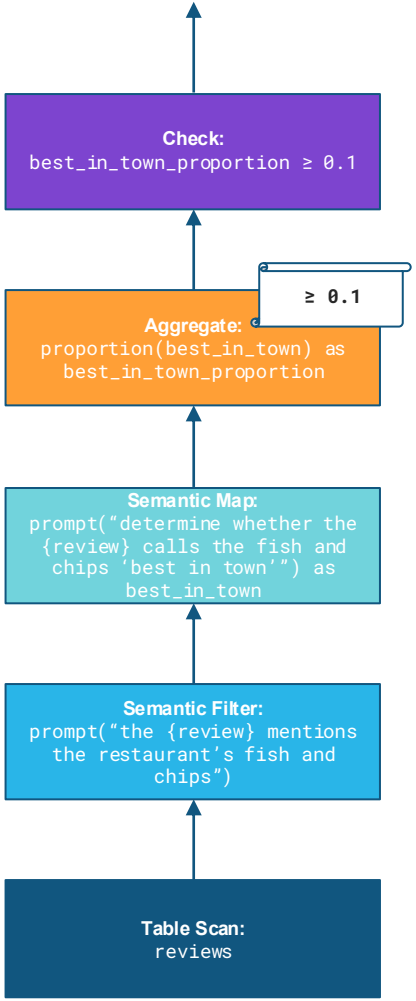
Semantic operator fusion

Multi-claim verification

Optimizations: Early Stopping

Stop aggregating once the downstream predicate is resolved

- Existential (`bool_or`): stop if **witness** is seen
- Universal (`bool_and`): stop if **counterwitness** is seen
- Cardinal (`count_if`) & proportional (`proportion`): **push** comparison predicates into aggregations

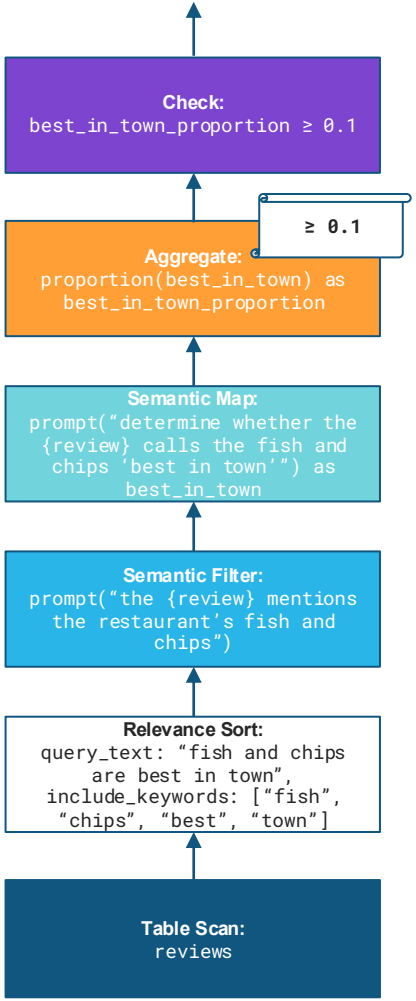


Optimizations: Relevance Sorting

Encourage early stopping by **sorting** tuples wrt **relevance**

Useful for quantifiers with small thresholds

1. LLM analyzes quantifier & predicate
2. LLM generates query text & search keywords
3. Query text → query vector
4. Rank tuples based on semantic & syntactic similarity
5. Use Reciprocal Rank Fusion (RRF)[1] to combine rankings



[1] Cormack et al., SIGIR (2009)

Optimizations: Estimation with CIs

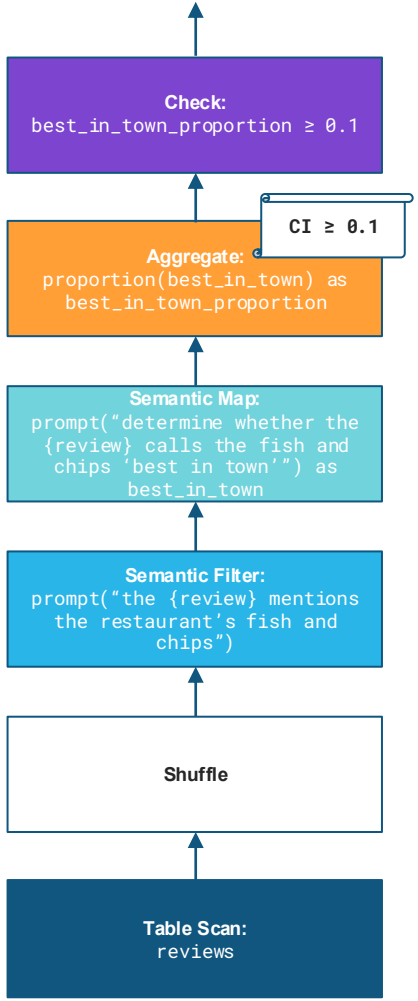
Estimate larger counts & proportions with **anytime-valid CIs**[1]

Insert **shuffle** operator to guarantee statistical validity

Stop once comparison can be determined with CI

E.g., entire CI is above or below threshold

Apply **Bonferroni correction** for multiple estimates



[1] Waudby-Smith and Ramdas, Journal of the Royal Statistical Society Series B (2024)

Provenance for Verification Queries

A query returns:

- **Verification result:** whether the claim is true or false
- **Provenance:** an explanation of how the result depends on facts in the input

Provenance based on **semiring semantics** for first-order logic[1]

with extensions (e.g., for cardinal and proportional quantifiers)

Provenance annotations are **propagated** through a query with tuple attributes

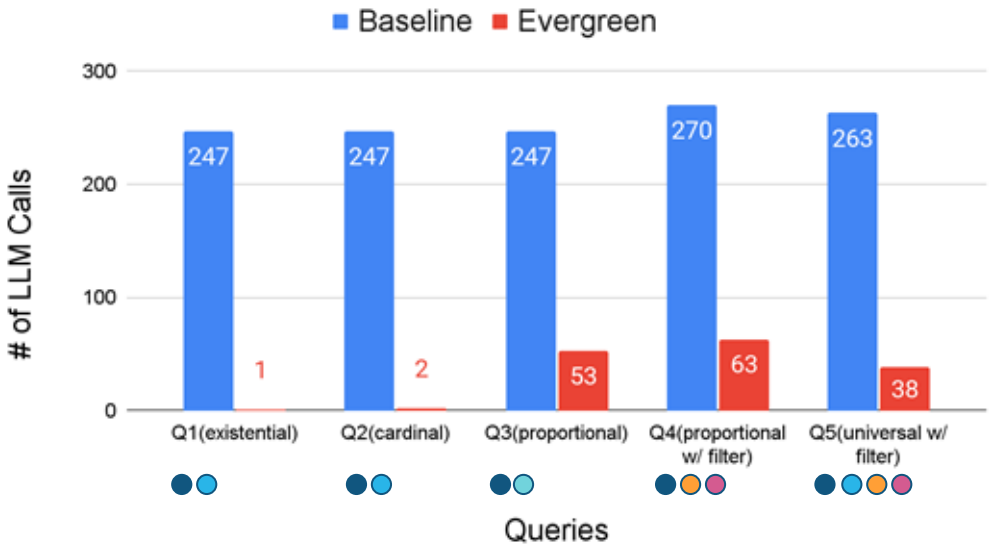
[1] Erich Grädel and Val Tannen, Model Theory, Computer Science, and Graph Polynomials (2025)

Preliminary Evaluations

Baseline: unoptimized semantic query processing engine

Evergreen: optimized semantic query processing engine

- Early stopping
- Relevance sorting
- Estimation w/ CIs
- Radius filtering
- Semantic operator fusion



Conclusions & Ongoing Work

By treating claims as **semantic verification queries**, Evergreen enables **reliable** and **efficient** claim verification for semantic aggregates

Ongoing work:

- Cost-based optimizations
- Integration with Snowflake's Cortex AISQL