VectraFlow: An AI-Augmented Data-Flow System

Shu Chen, Alexander Lee, Duo Lu, Deepti Raghavan, Malte Schwarzkopf, Uğur Çetintemel

BROWN

1. VectraFlow

A **data-flow engine**

that natively supports modern **ML models** with

an **extended relational model** for **unstructured** and

multi-modal data processing

Supports **stream** and **batch** processing

2. Lighthouse Domain: Medical Data Lakes

(collaboration with the RI Hospital)

Example apps:

- Medical data summarization
- Early warning system
- Compliance monitoring
- Automatic report generation

Key requirements:

- Integrate ML models (including LLMs)
- Support stream and batch oriented processing
- Ensure high reliability and scalability

3. Data and Query Model

Classical **data-flow** architecture with an **extended** relational model:

- **Data types**
  - Vector (sparse and dense)
  - Unstructured (e.g., free-form text, images)
- **Manipulation operators**
  - E.g. convert data to vectors, cluster vectors
- **Semantic relational operators**
  - Based on vectors, LLM prompts, and general ML models
  - Retain general semantics of relational operators

4. Example Semantic Operators

- **iV-Filter()**: applies **embedding similarity** to **select** incoming tuples (Lu et al., 2025)
  - E.g., **identify** incoming patient records that are similar to historical patient records
- **P-Agg()**: prompts an **LLM** to **aggregate** over a window of tuples (Patel et al., 2024)
  - E.g., **summarize** over multiple medical documents
- **M-Filter()**: invokes a **classifier** to **select** tuples based on their attributes (Lu et al., 2025)
  - E.g., **identify** abnormalities in medical imaging

5. iV-Filter (Lu et al., 2025)

Each base vector has a **radius**

iV-Filter: selects **input vectors** that fall within the **radii** of **base vectors**

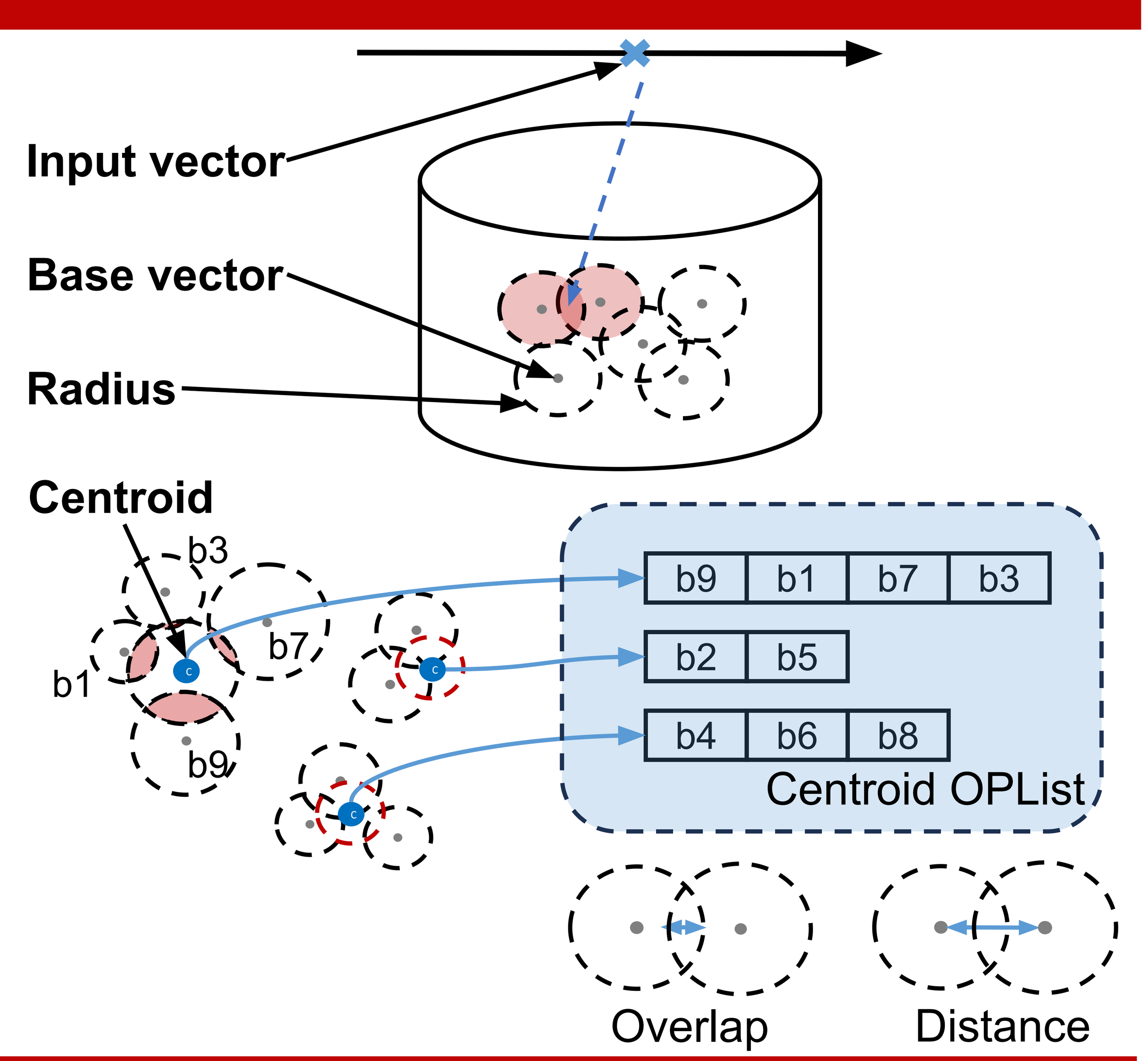
and returns the corresponding base vector IDs

Centroid OPList (Overlapped Partition List):

- Insight: base vectors containing the incoming vector must **overlap**
- OPList: **list** of base vectors that **overlap** with the given base vector
- Centroid OPList**: cluster** base vectors and assign a **radius** + **OPList** to each **centroid**

Search: assign input vector to the nearest centroid and **scan** its **OPList**

Other optimizations: batching, sorting, bucketing, early stopping



6. Semantic Integrity Constraints

Problem: semantic operators may yield **erroneous** results

Solution: **guardrails** around semantic operators to enforce **data consistency**

User-specified **predicates** on output tuples

Can apply **constrained decoding** for certain predicates

Otherwise,

if tuple violates predicates, **retry** operator

if specified retry threshold is reached, **drop** tuple

7. Integrity Constraint Classes

IC Class	Use Case
Domain	Medication dose stays within clinically safe **boundary**
Inclusion/exclusion	Generated business report doesn't **contain** undesirable language
Grounding	Extracted test records are **present** in the original medical document
Check <predicate>	Evaluate **arbitrary** predicates (e.g., simple statements, UDFs)

8. Enforcing Grounding Constraints

Want: attribute value is **grounded** in its **source tuple(s)**

- Recursively apply **checks** to all attributes in the attribute's **lineage**
- **Check**: output value is **grounded** in input value(s)
- Require different **grounding semantics** depending on the **use case**

Semantics	Verification Mechanism	Use Case
Match	Exact keyword match	Extractive
Similarity	Similarity score	Abstractive
Model	LLM evaluator	Extractive + abstractive

9. Acknowledgements

Special thanks to other members of the team who contributed to this work:
Justin Chan, Simeng Feng, Michael Fu, Nicolas Kim, Evan Li, Akshay Mehta, Weili Shi, Franco Solleza, Jonathan Zhou