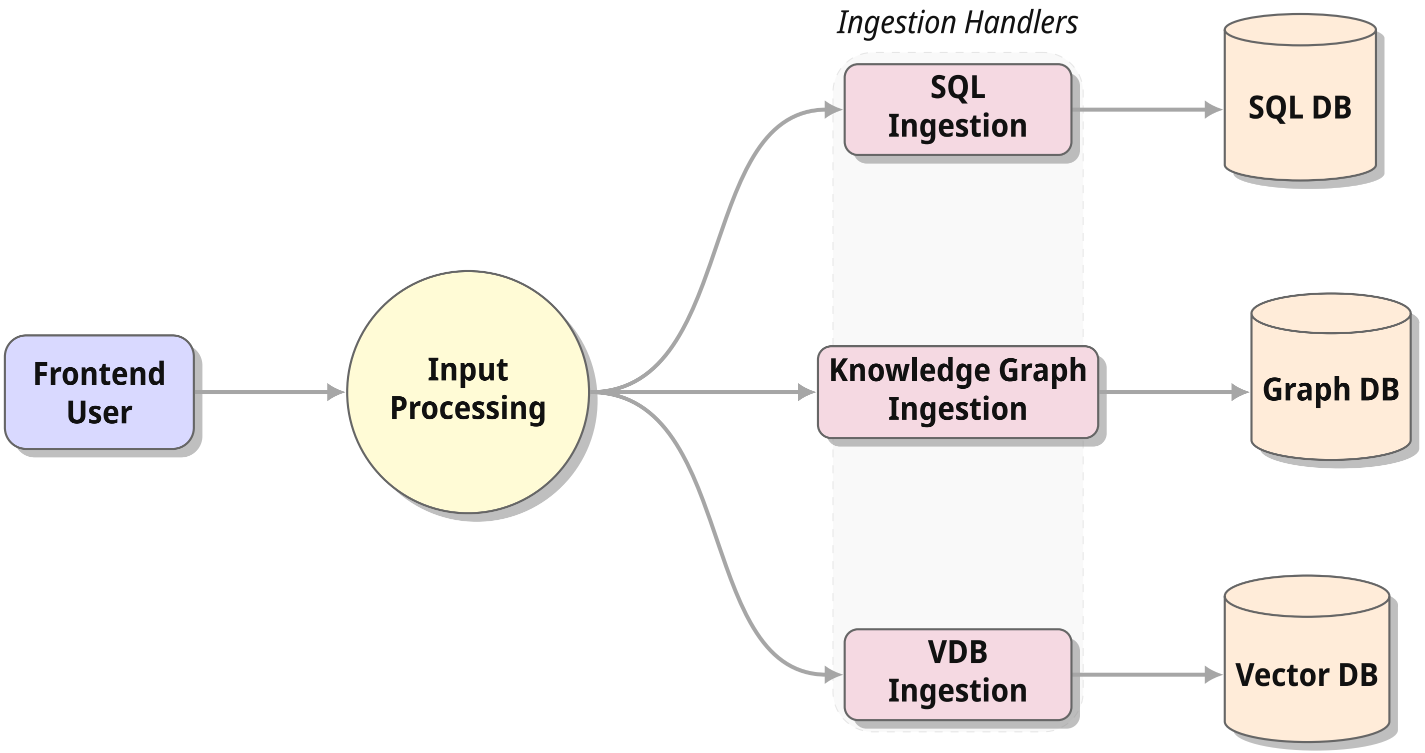
KURSE

*built entirely with open source models and infrastructure

What is KURSE?

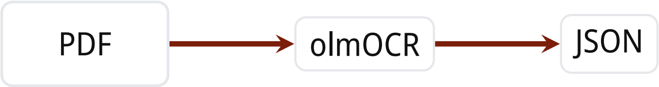
- Neuro-Symbolic Inference Engine
- Intelligent tri-store architecture: SQL, Vector DB, and Knowledge Graph
- Ontology-driven structure enabling hierarchical reasoning for events & relations

Let’s see how Kurse works


Ingestion Handlers
Frontend User
Input Processing
SQL Ingestion
Knowledge Graph Ingestion
VDB Ingestion
SQL DB
Graph DB
Vector DB

Input Processing

- Takes any unreadable data such a handwritten documents, pdfs and converts them into a readable format
- **Engine:** Powered by **olmOCR** (based on Qwen 7B)
- **Goal:** Structured JSON output optimized for LLM readability
- **Complex Element Handling:**
  - Tables & Complex Structures → HTML
  - Standard Text → `Markdown`



Input Processing

- **Metadata Injection:**

  `document_id`, `page_no`, `char_count`

- **Performance Metrics:**
  - Latency: ∼1.2s per page
  - Supports Parallel Batch Ingestion

**Metrics based on local running with a 4090 GPU*

Vector Database Ingestion

- **Model Architecture:** Uses **CLIP**
  - Creates a **shared vector space** for both images and text
  - Enables seamless cross-modal search and retrieval
- **Embedding Specification:** High-fidelity **1024** Dimensions
- **Stored Metadata Payload:**
  - `document_id`, `chunk_id`, `page_no`
  - `source_file`, `text_preview`
  - `table_no`, `image_no`

Knowledge Graph

- Nodes connected by edges (relations)
- **Structure:**

  `Node(type, props)` — `Relation(type, origin, version)` — `Node(type, props)`

- **Example:**

  `(Person: "John")` — WORKS_AT(src: doc_01, v: 1.2) — `(Org: "Acme Corp")`

Node Types

> **Entities** — concrete things: products, persons, organizations

> **Properties** — descriptive attributes: material, style, designation

> **Literals** — terminal values with no relational value: price, date

N-ary Relations

- Relations involving **3+ participants** that can’t be reduced to binary pairs
- **Why important:** Complex events need a central node to capture context (time, location, method) that belongs to the event itself, not individual actors

Example:

```
(Person: "Marcus") –[COMMITTED]– (Crime: "Heist_NYC_2024")
–[LOCATION]– (Place: "Manhattan")
–[WITH]– (Person: "Sofia")
–[ON_DATE]– (Date: "2024-03-15")
```

What is an Ontology?

- A predefined set of **things** with properties & relations

Entity (Class)

```
schema:Person a rdfs:Class ;
  rdfs:label "Person" ;
  rdfs:subClassOf schema:Thing .
```

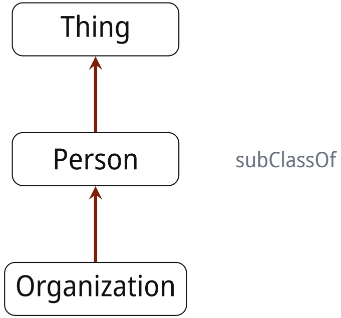
Relation (Property)

```
schema:worksFor a rdf:Property ;
  domainIncludes schema:Person ;
  rangeIncludes schema:Organization .
```

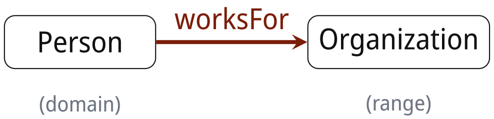
Source: Schema.org ontology

Hierarchy & Reasoning

Class Hierarchy



Inferred Type Constraints



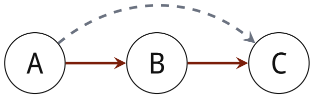
If x `worksFor y`:
⇒ x must be a **Person**
⇒ y must be an **Organization**

Ontologies enable automatic type inference via first-order logic

Axioms & Inference

Axioms map to first-order logic, enabling inference of **implicit facts** from explicit data

Transitivity

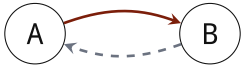


`partOf`

A partOf B, B partOf C

⇒ A partOf C

Symmetry

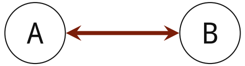


`knows`

A knows B

⇒ B knows A

Equivalence



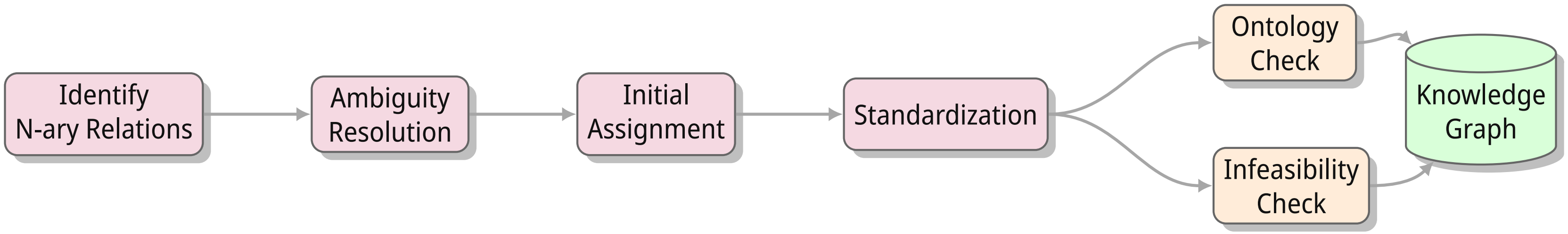
`sameAs`

A sameAs B

⇒ $A \equiv B$

Example: Given John `worksFor` Acme and `worksFor` has domain `Person`
⇒ Infer: John is a `Person` (implicit fact)

KG Ingestion Pipeline



N-ary Event Identification

- LLM identifies **complex events** with 3+ participants
- Events that can’t be reduced to simple binary relations

Manifest Storage

```
{
 "short_id": "mtg_rome_01",
 "description": "Meeting
  between Victor & Maria
  in Rome..."
}
```

Event VDB

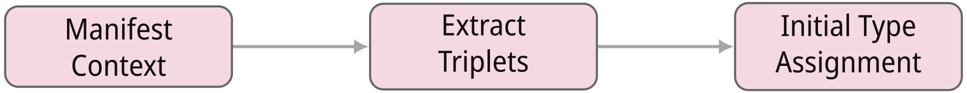
Semantic search for existing events

Prevents duplicate event creation

Returns top-k similar events

Events are reused across chunks via VDB similarity matching

Ambiguity Resolution & Initial Assignment



- **Manifest Mandate:** LLM must use exact `short_id` for identified events
- **Initial Assignment:** LLM infers types (Person, Organization, Event...)

Ontology & Standardization

Types VDB

`rdfs:Class` entries

Person, Organization,
Location, Event...

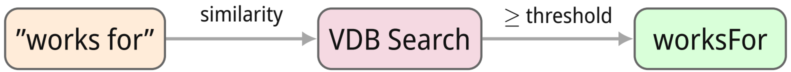
+ label, superclass

Relations VDB

`rdf:Property` entries

worksFor, locatedIn,
memberOf, knows...

+ domain, range



”works for”

similarity

VDB Search

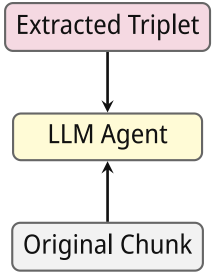
$\geq$ threshold

worksFor

Raw terms kept if similarity too low (no hallucination)

Verification & Validation

Chunk Verification



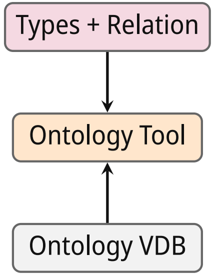
Extracted Triplet

LLM Agent

Original Chunk

Supported? ✓ Keep
Fabricated? × Drop

Ontology Feasibility



Types + Relation

Ontology Tool

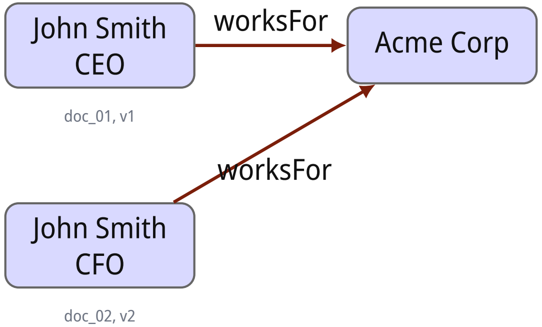
Ontology VDB

Valid combo? ✓ Keep
Infeasible? × Drop

Double-check: both agent and tool must approve before KG insertion

Conflict Resolution & Versioning

- **Problem:** Same entity, different information across sources



Solution: New node with different metadata

Tracks: `document_id`, `chunk_id`, `page_no`, `ingested_at`

Preserves conflicting information with full provenance

Beyond Storage

KURSE: An Intelligent Data Understanding & Storage System

- Highly adaptable architecture for diverse domains
- Neuro-symbolic reasoning enables:
  - Constraint Satisfaction
  - Graph Traversal
  - Logical Inference

Let’s see some applications...

Application: Legal Contradiction Detection

- **The Idea:** Ingest case files (emails, receipts, testimonies) to verify if a suspect’s alibi is physically possible

- **Why KURSE Fits:**
  - Treats this as a **Constraint Satisfaction Problem**
  - Maps entities to `Location(Time)` states
  - Applies rule: `Entity cannot be at A and B at Time T`
  - Catches contradictions RAG misses (e.g., ”at home” vs receipt at bar)

- **Required Changes:** Change the Agentic Search’s Query Answering agent to find conflicts between given info and retrieved info, and break the query into sub-queries based on the events claimed to happen in the input

Application: Market Risk Identification

- **The Idea:** Predict how localized events (e.g., Thailand flood) impact portfolio assets via supply chain effects
- **Why KURSE Fits:**
  - Treats this as a **Graph Traversal Problem**
  - Builds supply chain: `Flood` → `Factory` → `Supplier` → `Apple`
  - Propagates risk through transitive reasoning
  - Finds 2nd/3rd order effects that keyword search misses
- **Required Changes:** Add multi-hop traversal with risk propagation. Again, a minor change to the part of the Agentic Search that creates the sub-quries.

BeeKurse

What is BeeKurse?

Conversational E-commerce Platform

Excelling in product recommendations

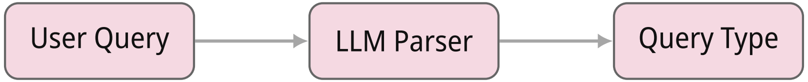
- **OCR + Reasoning for Small Vendors:**
  - Update inventory by writing on paper daily
  - No need for detailed catalogued updates
- **VDB/KG Structure Enables:**
  - Smart product recommendations
  - Product comparisons

Let’s discuss the changes we made to KURSE

Adapting KURSE for Commerce

- **Dropped** rigid ontology system
- **Replaced with** VDB seeded with relation types
- **Why?** Commerce needs:
  - Creative, particular relations
  - Less rigid reasoning
  - Flexibility over formality

Search Algorithm: Query Classification



5 Query Types:

- **SEARCH** — Product discovery
- **DETAIL** — Product info Q&A
- **CHAT** — Conversational
- **CART_ACTION** — Add/remove items
- **CART_VIEW** — View cart/wishlist

Query Type: SEARCH — Parsing

What it is:

- User wants to **find new products**
- Most complex query type

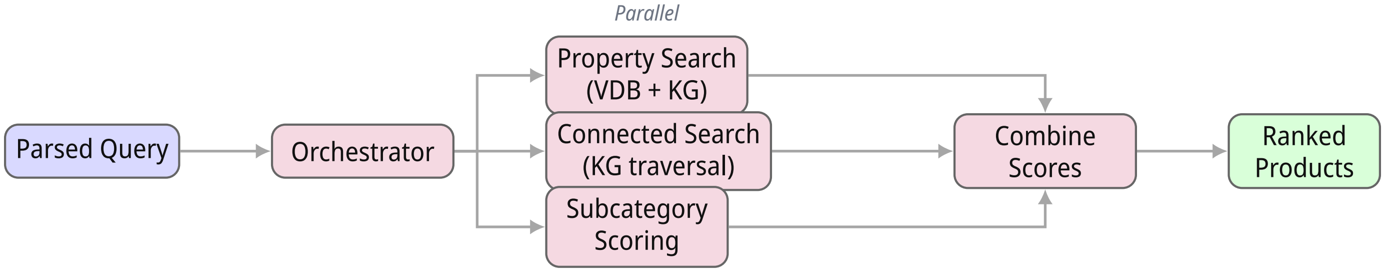
Example:

`"Red cotton shirt under $30"`

Parsed Output:

```
{
 "query_type": "SEARCH",
 "category": "clothing",
 "subcategory": "shirt",
 "properties": [
  ["red", 1.5, "HAS_COLOUR"],
  ["cotton", 1.2, "HAS_MATERIAL"]
 ],
 "literals": [
  ["price", "<", 30.0, 0.1]
 ]
}
```

Query Type: SEARCH — After Parsing



- **Property Search:** VDB similarity + KG relation matching
- **Connected Search:** Find related products via KG edges
- **Combine:** Weighted sum of all scores → rank → return top-K

Query Type: DETAIL — Parsing

What it is:

- User asks about **specific product(s)**
- Can reference by ID or short code
- Supports comparisons

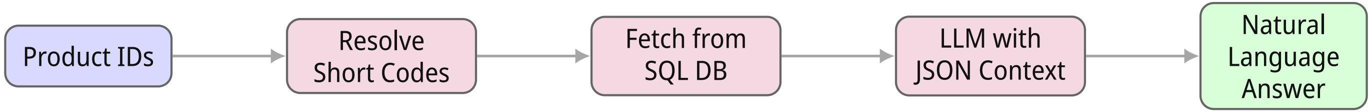
Example:

`"Compare the materials of p-123 and p-456"`

Parsed Output:

```
{
 "query_type": "DETAIL",
 "product_ids": [
  "p-123",
  "p-456"
 ],
 "original_query":
  "Compare the materials"
}
```

Query Type: DETAIL — After Parsing



- **Resolve:** Short codes (e.g., ”44QM”) → full product IDs
- **SQL Fetch:** Retrieve complete product metadata
- **LLM:** Product data as JSON context → answer user’s question

Query Type: CHAT

What it is:

- Greetings, general questions
- No product search needed

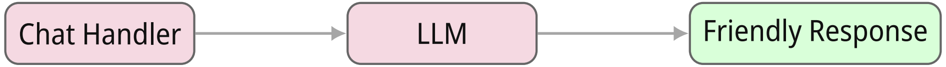
Example:

`"Hello!"`, `"How does this work?"`

Parsed Output:

```
{
 "query_type": "CHAT",
 "message": "Hello!"
}
```

After Parsing:



No database queries — direct LLM response

Query Type: CART_ACTION

What it is:

- Add/remove items from cart
- Manage wishlist

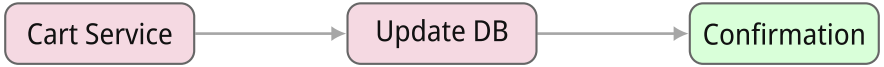
Example:

`"Add p-123 to my cart"`

Parsed Output:

```
{
  "query_type": "CART_ACTION",
  "action": "add",
  "target": "cart",
  "product_ids": ["p-123"]
}
```

After Parsing:



Cart Service → Update DB → Confirmation

Query Type: CART_VIEW

What it is:

- Display cart or wishlist contents
- Show product details

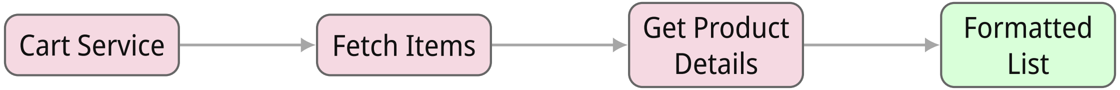
Example:

`"What's in my cart?"`

Parsed Output:

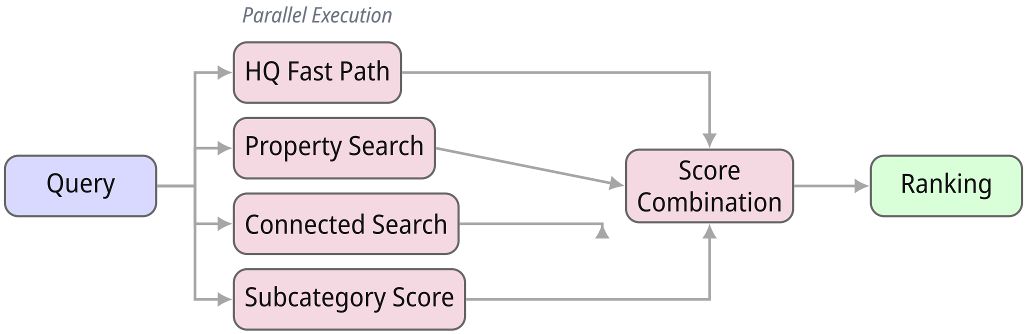
```
{
 "query_type": "CART_VIEW",
 "target": "cart"
}
```

After Parsing:



Cart Service → Fetch Items → Get Product Details → Formatted List

Search Orchestration Pipeline



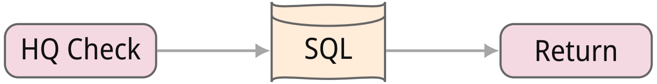
Four parallel scoring paths, each contributing to final product ranking

- **HQ:** Fast path for repeat purchases
- **Property Search:** Semantic + KG matching
- **Connected Search:** Recommendations via graph
- **Subcategory:** Type-matching bonus

HQ Fast Path (Hurry Query)

What it is:

- Direct SQL lookup for exact product
- Bypasses all other search paths
- Returns immediately if found



When applicable:

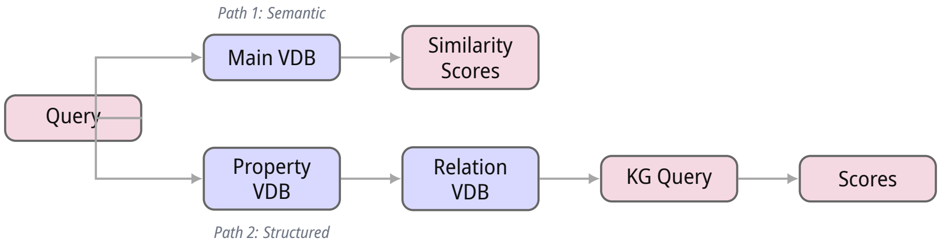
- `is_hq = True` (parser flag)
- `prev_productid` exists
- E.g., ”my usual order”

Why we use it:

- Instant response for repeats
- Skips expensive VDB/KG queries
- Best user experience

Property Search (RQ — Relevance Query)

What it is: Main search path using VDB + KG



When applicable:

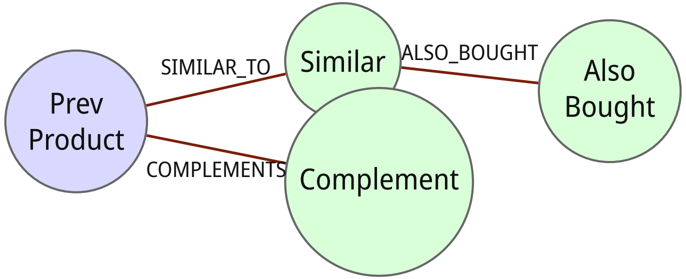
- Always runs for SEARCH queries
- (Unless HQ succeeds first)

Why we use it:

- Semantic: ”crimson” ≈ ”red”
- KG: structured property matching

Connected Search (SQ — Similarity Query)

What it is: KG graph traversal to find related products



When applicable:

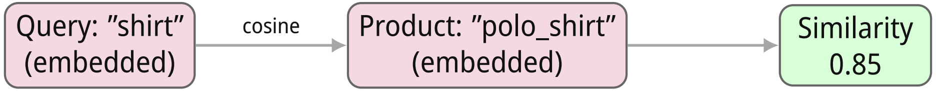
- `prev_productid` provided
- `prev_storeid` provided
- ”Show me similar to this”

Why we use it:

- Recommendations from patterns
- Store loyalty bonus (+0.2)
- Connected bonus (+0.5)

Subcategory Scoring

What it is: Cosine similarity between query and product subcategories



When applicable:

- Always runs
- Adds bonus if similarity $>$ 0.5

Why we use it:

- ”shirt” ranks shirts $>$ pants
- Even if pants match ”red cotton”
- Up to +0.4 bonus

Property Weighting

Weight Scale:

0.5 = Nice-to-have
1.0 = Normal importance
1.2-1.5 = Important
2.0 = Critical / must-have

Formula:

```
score = similarity × weight
```

Example: "red cotton shirt"

```
Properties:
  "red" weight=1.5 (important)
  "cotton" weight=1.0 (normal)

Product A (red shirt):
  red_sim=0.95 → 0.95×1.5 = 1.43
  cotton_sim=0.80 → 0.80×1.0 = 0.80

Product B (maroon shirt):
  red_sim=0.70 → 0.70×1.5 = 1.05
  cotton_sim=0.85 → 0.85×1.0 = 0.85
```

Score Combination & Final Ranking

Final Score Formula:

```
final_score = Σ(property_scores) + connected_bonus + subcategory_bonus -
literal_penalties
```

Bonuses

Connected: **+0.5**
Same store: **+0.2**
Subcategory: **up to +0.4**

Literal Penalties

Buffer zone: 10%
$31 on $30 limit:
small penalty (-0.05)
$35 → filtered out

Superlatives

”cheapest”, ”best rated”
70% relevance
30% literal value

Example: Product A: props=2.23 + connected=0.5 + subcat=0.34 - penalty=0 = **3.07**

Thank You

Contributions

- **Preprocessing, OCR & Parsing**
  Abhinav Goyal
- **Dockerization**
  Kunjan Manoj
- **Knowledge Graph Ingestion**
  Varshith Kada, Siripuru Abhiram
- **VDB Ingestion**
  Siripuru Abhiram, Varshith Kada
- **SQL Ingestion**
  Vishnu Teja
- **Search**
  Varshith Kada
- **Strontium Agent**
  Varshith Kada, Himesh
- **Frontend for Vendors**
  Himesh
- **Backend**
  Vishnu Teja
- **Synthetic Data Preparation**
  Bhuvan