# How Neural DB works

Natural Language Query → Answer, without explicit schematizing.

# How Neural DB works

- Goal is to go from NL query all the way to an answer
- Cost and latency should be low — every deterministic step should be in Python, only calling an LLM for decisions that can’t easily be written in code.

# How Neural DB works — the seven step path

1. Understanding the query
2. Finding sources via ARD
3. Choosing tables
4. Resolving entities & measures
5. Planning the requests
6. Fetching the data
7. Validate and compose answer

- There will be errors … allow backtracking to any stage

# How Neural DB works

**What percentage of people in Detroit live below the poverty line?**

**Answer: Detroit city, Michigan = 31.5 percent (2022)**

# Demo: Understanding the query

```
Question structure: a single value
entity: Detroit
measure: percentage below the poverty line
1 fallback reading held in reserve
a ratio: numerator people that live below the poverty line of Detroit
denominator people of Detroit, form percent
```

# Typed queries

| Type | Fields | Example |
|---|---|---|
| `EntityValueQuery` | entity, attribute | India's GDP |
| `MultiEntityValueQuery` | entities (2), attribute | Unemployment in Ohio and Texas |
| `MultiEntityValueQuery` | entity, attributes (2) | Population and income of Detroit |
| `RatioQuery` | numerator, denominator | SNAP households share |
| `RankingQuery` | population, attribute | Highest poverty rate |
| `BreakdownQuery` | entity, dimension | Arizona by race |

# The model panel

- Example: gemini-3.8-flash and qwen3.8-27b
- A third is asked if they differ.

# Stage 2. Finding sources via ARD

```
query: POST ard.neuralweb.dev/search
text: Find tables for: percentage of people that live below the poverty line Detroit. Background only ...
texts: ["percentage of people that live below the poverty line Detroit", "What percentage of people in Detroit live below the poverty line?", "percentage of people that live below the poverty line"]
```

| Score | Source | Table | Title |
|---|---|---|---|
| 100 | census | `dp03-0128pe` | % Below the poverty level, all people |
| 88 | datacommons | `count-person-belowpoverty…` | Population Below Poverty Level Status in Past Year |
| 100 | census | `dp03-0128e` | Below the poverty level, all people |
| 100 | census | `dp03-0134pe` | All people, 18 to 64 years |
| 100 | census | `dp03-0133e` | All people, 18 years and over |
| 100 | census | `dp03-0133pe` | % All people, 18 years and over |
| 80 | census | `dp03-0137pe` | % All people, unrelated individuals 15 years and over |
| 80 | census | `dp03-0129e` | All people, under 18 years |
| 80 | census | `dp03-0136e` | All people, people in families |
| 80 | census | `dp03-0136pe` | % All people, people in families |

- One search for each piece of data.

# Models as a source of last resort

- Example: which states border Canada.
- Models list the members; sources supply every number.

# Plugins

- Commonalities between the tables from any source.
- A plugin is the code for one source.
- Can handle source specific idiosyncrasies
- Default plugin
- census, datacommons, sec, worldbank, …

# Demo: Choosing source + tables

- Before a source may choose tables, its plugin determines whether the API can serve that kind of query. E.g., IRS 990 tables are indexed only one way. Census can’t do top-N.

# How entities are resolved

- Each source has its own identifier scheme
- Data Commons: resolve API, then a model
- Census: place list, then a model
- SEC: company to CIK
- Census: Detroit is state:26/place:22000
- Data Commons: Detroit is geoId/2622000

# Resolving measures

- This is done by ARD (the ARD entry should contain the variable name)
- Every big source has its own arbitrary variable names (how many ways can you say ‘earnings’).
- Census -> DP03_0128PE (percent)
- Census -> DP03_0128E
- Data Commons -> sdg/SI_POV_NAHC

# Planning the requests

```
Census: api.census.gov/data/2022/acs/acs5/profile?get=NAME,DP03_0128PE&for=place:22000&in=state:26
Data Commons: variable sdg/SI_POV_NAHC, place geoId/2622000
```

- Deterministically construct queries
- Construct queries for all viable sources

# Issuing queries

- Issue queries one by one
- Stop when we get an answer

# Final answer

- Written from returned answers by deterministic code
- Text + charts

# Test layers

- Unit tests: NeuralKG2 has 228 pytest tests in 11 files in tests/, and ARD has 131.
- Golden files: 97 questions in each of 10 per-stage files, about 970 stage-level checks. Understanding has its own set of 100.
- End-to-end sets: test100.json has 100 questions. cross-source.json has 3, evaluation.json has 8 and live-config.json has 3.

# Testing and evaluation

```
No health insurance in Michigan -> 68,155
Share with a bachelor's degree in Texas -> 6,669,107
```

- A complete answer can still be wrong. Need human eval for golden set

neuralkg2: tools/evaluate_examples.py neuralkg2: tools/collect_examples.py neuralkg2: docs/evaluations/