Snowflake Cortex AI

From GenAI concepts to AI Functions, Search, Analyst, Agents and CoWork

Presenter



/arifmazumder

Mohammed Arif, PhD
Gen AI Solution Architect

Big Data | Machine Learning | AI

Mohammed Arif has more than eighteen (18) years of working experience in Information Communication and Technology (ICT) industry. The highlights of his career are more than seven (7) years of holding various senior management and/or C-Level and had five (5) years of international ICT consultancy exposure in various countries (APAC and Australia), specially on Big Data, Data Engineering, Machine Learning and AI arena.

He is also Certified Trainer for Microsoft & Cloudera.

Our learning path

01
Setup and big picture
Trial, ecosystem map, setup

02
GenAI
AI, GenAI, LLM, transformers

03
AI Agents
Agentic, SKILL.md, create one

04
Semantic views
SQL and YAML formats

05
MCP and harness
How agents connect and drive

06
Cortex Analyst
Questions to SQL, step by step

07
Cortex Search
Search by meaning, step by step

08
AI Functions
13 functions and 3 helpers

09
Together, at scale
CoWork, quotas, CoCo

Can we run this on a trial account?

Feature	Trial account	Note
Cortex AI Functions	Yes, capped	About 10 credits per day without a card
Cortex Search, Analyst	Yes	Keep data small, use an XS warehouse
Cortex Agents, CoWork	Yes	Turn on cross-region inference
CoCo in Snowsight	Yes	Needs cross-region inference
CoCo CLI	Separate trial	Own sign-up page for the CLI
Per-user quotas	Yes	Admin feature, Budgets tab

Sign up: **Enterprise edition**, **AWS US West (Oregon)** for the widest model choice. Corporate accounts work the same, without the daily cap.

01 · SETUP

Snowflake AI Ecosystem

One platform, four layers

CoWork · business users
Ask questions at ai.snowflake.com

CoCo · builders
AI coding agent for SQL, Python, apps

Cortex Agents · plan, pick a tool, answer

AI Functions
AI inside SQL

Cortex Search
Text and documents

Cortex Analyst
Questions to SQL

MCP Server
Tools for outside apps

Your governed data: tables, stages, documents

RBAC · masking · per-user AI quotas

Redrawn from Snowflake's official AI overview: docs.snowflake.com/en/guides-overview-ai-features · CoWork was Snowflake Intelligence, CoCo was Cortex Code

Hands-on: set up the account

1. Sign in to Snowsight

2. Left menu: **Projects » Workspaces**

3. **+ Add new » SQL File**, name it **00_setup**

4. Paste the code, then **Run all**

5. Check: **SHOW WAREHOUSES;**

```
USE ROLE ACCOUNTADMIN;
CREATE WAREHOUSE IF NOT EXISTS cortex_lab_wh
  WAREHOUSE_SIZE = 'XSMALL' AUTO_SUSPEND = 60;
CREATE DATABASE IF NOT EXISTS cortex_lab_db;
CREATE SCHEMA IF NOT EXISTS cortex_lab_db.retail;

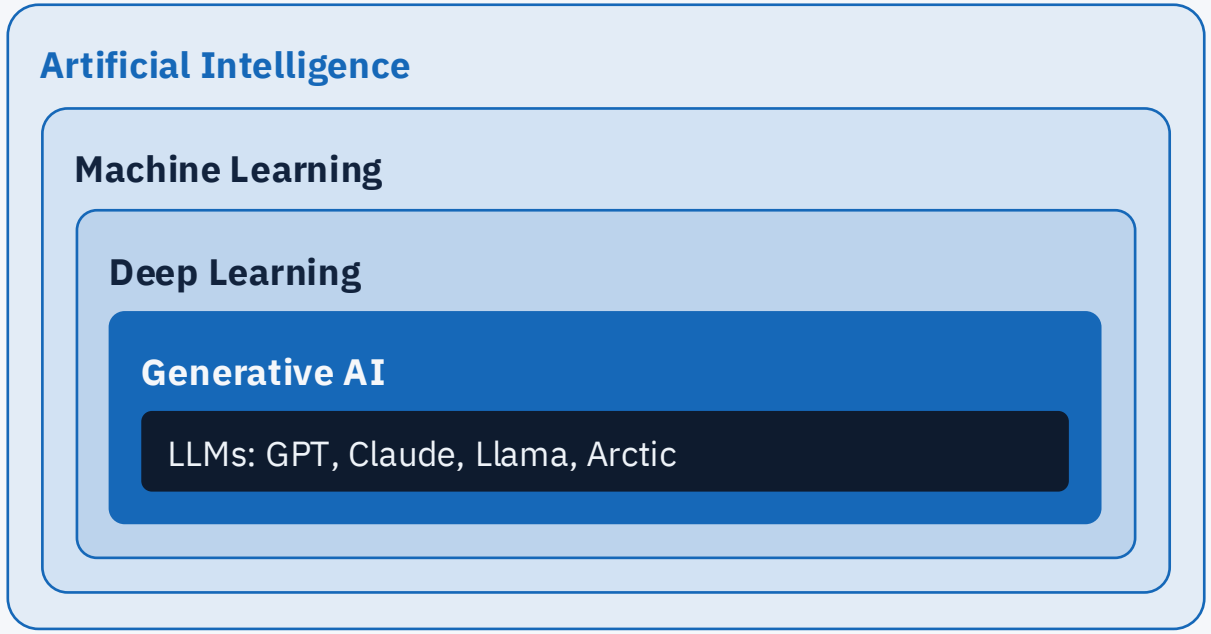
-- reach models hosted in other regions
ALTER ACCOUNT SET CORTEX_ENABLED_CROSS_REGION = 'ANY_REGION';

-- PUBLIC has CORTEX_USER by default; a lab role
CREATE ROLE IF NOT EXISTS cortex_lab_role;
GRANT DATABASE ROLE SNOWFLAKE.CORTEX_USER
  TO ROLE cortex_lab_role;
GRANT ROLE cortex_lab_role TO USER <your_user>;
```

02 · GENAI

GenAI

AI is a family of nested ideas



AI: machines doing tasks that need human-like thinking.

ML: learns patterns from data instead of fixed rules.

Deep Learning: ML with many-layer neural networks.

GenAI: deep learning that **creates** new content.

GenAI creates new content



Generative AI is a type of AI that **creates new content**, such as text, images, video, audio and code, based on patterns it learned from existing data.

G

Generative: makes new output

P

Pre-trained: learned on huge text first

T

Transformer: the network design

02 · GENAI

LLM

Large Language Model

An LLM predicts the next token

1 · PROMPT

“Snowflake is a”

2 · TOKENS

`Snow` `flake` `is` `a`

3 · MODEL SCORES NEXT WORDS

cloud **62%**

data 21%

white 9%

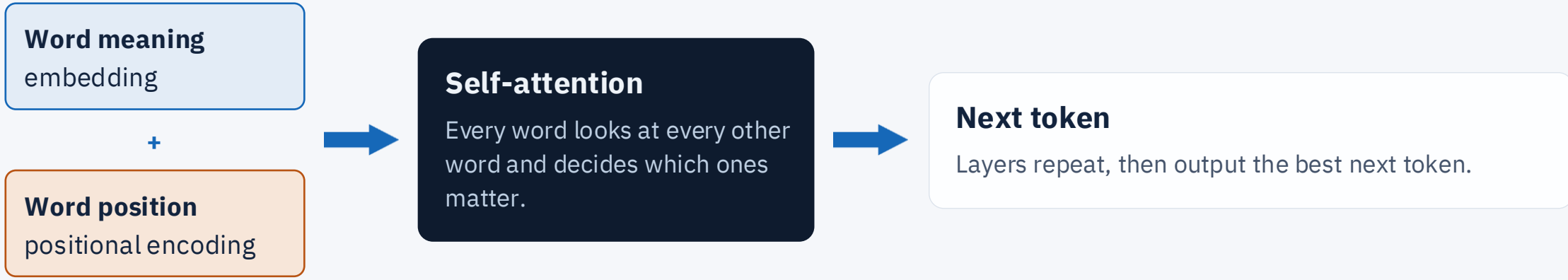
4 · Pick and repeat

Add “cloud”, then predict again, one token at a time, until done.

Why it matters for cost

1 token ≈ 4 characters. Cortex bills input and output tokens.

Transformers and positional encodings



Why position matters

“Customer refunded the store” and “Store refunded the customer” use the **same words**. Only positional encoding tells the model **who did what**.

03 · AI AGENTS

Agentic AI vs AI Agents

An agent is a thing. Agentic is a behaviour.

	AI Agent	Agentic AI
What	One unit: LLM + tools + instructions for a task	A system that plans, acts, checks and repeats toward a goal
Control	Answers a request, uses tools when asked	Breaks the goal into steps by itself, little human input
Scope	Narrow, one job	Multi-step, often several agents together
Snowflake	A Cortex Agent “Retail AI” with Analyst + Search	CoCo planning and building a full pipeline

Simple test: **who decides the next step?** You, then it is an agent tool. The AI, then it is agentic.

SKILL.md: teach an agent a repeatable job

What a skill is

A folder with one **SKILL.md** file: a name, a one-line description, then step-by-step instructions. Scripts or examples can sit beside it.

How the agent uses it

- Reads only the name and description at first
- Loads the full steps when a task matches
- Same result every time, no long prompt

WHERE SKILLS LIVE IN SNOWFLAKE

CoCo

.cortex/skills/ in a project, or ~/.snowflake/cortex/skills/. Run with $skill-name

Cortex Agents

Agent skills, stored on a stage or in a Git repo, attached to the agent

CoWork

User skills: + » Skills » Create new, or chat to make one

Anatomy of a SKILL.md

```
---
name: weekly-feedback-report
description: Use when asked for the weekly
  customer feedback report or a review summary.
tools:
  - sql_execute
---
# Steps
1. Query cortex_lab_db.retail.reviews_enriched
   for the last 7 days.
2. Count reviews by sentiment and by topic.
3. List the top 3 complaints per product.
# Output
One small table, then 3 short insights.
```

Frontmatter

Between the --- lines. The description is the trigger: write when to use it.

tools

Optional. What the skill may use. In CoCo CLI the SQL tool is sql_execute.

Body

Plain Markdown: the steps, and what the answer should look like.

AI AGENTS

How to create a Cortex Agent

Two ways: click through Snowsight, or one SQL statement

Step by step in Snowsight

1. Left menu: **AI & ML » Agents**, then **Create agent**
2. Pick database and schema, object name **retail_ai**, display name **Retail AI**
3. Add a description and 3 sample questions users will see
4. **Tools** tab: **+ Add** beside Cortex Analyst, pick **Semantic view** and a warehouse
5. **Tools** tab: **+ Add** beside Cortex Search, pick the service, ID and title columns
6. **Orchestration** tab: model (auto) and when to use each tool
7. Response instructions: tone, length, “use charts”, currency
8. **Save**, then test in the chat preview and open the trace

Syntax: CREATE AGENT with two tools

```
CREATE OR REPLACE AGENT
  cortex_lab_db.retail.retail_ai
  PROFILE = '{"display_name": "Retail AI"}'
  FROM SPECIFICATION $$
models:
  orchestration: auto
instructions:
  response: "Be brief. Use charts."
  orchestration: "Numbers: SalesAnalyst.
    Complaints: TicketSearch."
  sample_questions:
    - question: "Revenue by region?"
```

```
tools:
  - tool_spec:
      type: "cortex_analyst_text_to_sql"
      name: "SalesAnalyst"
      description: "Sales by product, region"
  - tool_spec:
      type: "cortex_search"
      name: "TicketSearch"
      description: "Customer support tickets"
tool_resources:
  SalesAnalyst:
    semantic_view: "cortex_lab_db.retail.retail_sv"
  TicketSearch:
    search_service: "cortex_lab_db.retail.ticket_search"
    max_results: "5"
$$;
```

GIVING AI BUSINESS MEANING

Semantic View Semantic Model (YAML)

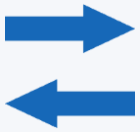
One business dictionary, two formats

It tells AI what your tables **mean**: how they join, what to group by, how to compute each number, and the words people use.

Tables + keys | Relationships | Dimensions | Facts, metrics | Synonyms | Verified queries

Semantic view

A Snowflake object in a schema
CREATE SEMANTIC VIEW in SQL
Roles and grants like a table
Query it with SEMANTIC_VIEW()
Recommended today



Semantic model (YAML)

A YAML text file
The older Cortex Analyst format
Easy to keep in Git and review
Converts to and from a view
Same parts, same meaning

Semantic views · Convert: SYSTEM$CREATE_SEMANTIC_VIEW_FROM_YAML and SYSTEM$READ_YAML_FROM_SEMANTIC_VIEW

Syntax: a semantic view

```
CREATE OR REPLACE SEMANTIC VIEW <name>
  TABLES ( <alias> AS <table> PRIMARY KEY (<col>), … )
  RELATIONSHIPS (
    <alias> (<fk_col>) REFERENCES <alias2> )
  FACTS ( <alias>.<name> AS <row expression> )
  DIMENSIONS ( <alias>.<name> AS <column>
    WITH SYNONYMS = ('…') )
  METRICS ( <alias>.<name> AS SUM(<column>) )
  COMMENT = '…'
  AI_SQL_GENERATION '<extra rules for Analyst>';
```

Order matters: FACTS before DIMENSIONS before METRICS

Need at least one dimension or metric

Query it with **SEMANTIC_VIEW()** in SQL

Cortex Analyst · Or build it with a wizard: AI & ML » Cortex Analyst » Create new » Semantic View

Syntax: the YAML semantic model

```yaml
name: retail_sm
tables:
  - name: orders
    base_table:
      database: CORTEX_LAB_DB
      schema: RETAIL
      table: ORDERS
    primary_key: {columns: [order_id]}
    dimensions:
      - name: region
        expr: region
        data_type: VARCHAR
        synonyms: [state]
    facts:
      - {name: revenue, expr: revenue, data_type:
NUMBER}
    metrics:
      - {name: total_revenue, expr: SUM(revenue)}
```

Also allowed: time_dimensions, relationships, verified_queries

YAML to a semantic view

```
CALL SYSTEM$CREATE_SEMANTIC_VIEW_FROM_YAML(
  'cortex_lab_db.retail',
  $$ <paste YAML> $$,
  TRUE); -- TRUE = check only
```

A semantic view to YAML

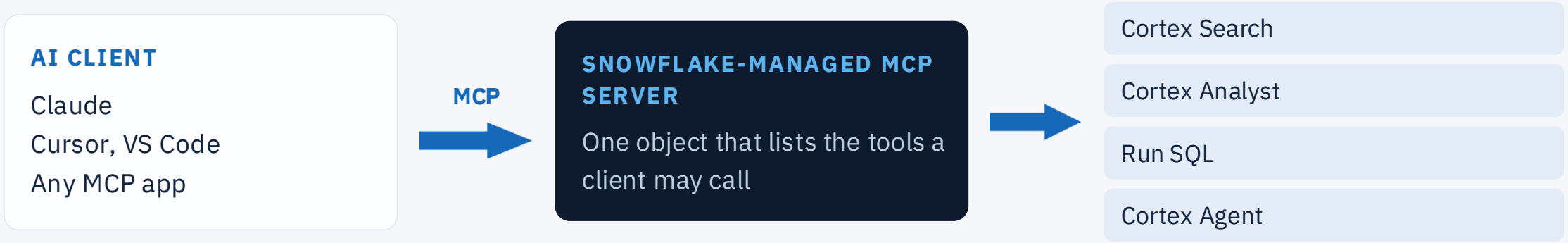
```
SELECT SYSTEM$READ_YAML_FROM_SEMANTIC_VIEW(
  'cortex_lab_db.retail.retail_sv');
```

05 · MCP

MCP Servers

Model Context Protocol

MCP is the USB-C port for AI tools



Without MCP
Custom code for every app and every tool

With MCP
One standard plug, governed by Snowflake roles

06 · AGENT HARNESS

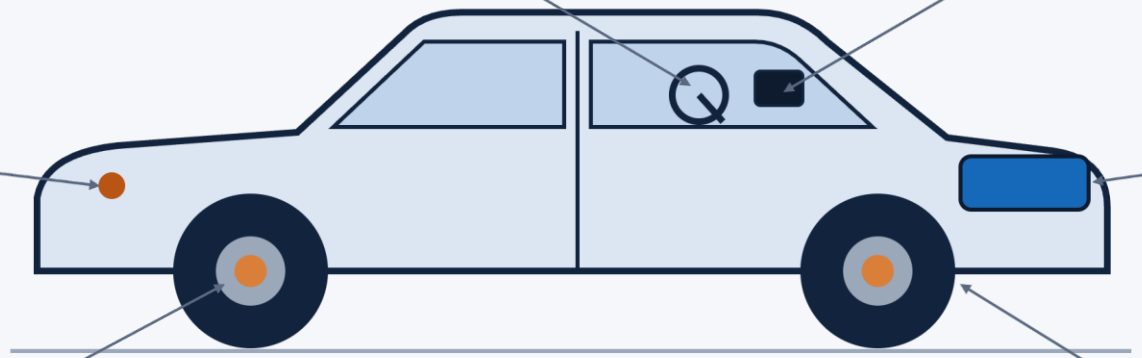
Harness

An engine alone does not drive

Steering = orchestration
Plan, act, check, repeat

GPS = context, memory
Semantic view, skills, chat

Fuel = data, tokens
Your governed tables



Engine = the LLM
Claude, Llama, Mistral

Brakes = guardrails
Roles, masking, quotas

Wheels = tools
SQL, Search, Analyst, MCP

Agent = engine (LLM) + harness (everything else). In Snowflake: Cortex Agents and CoCo.

07 · CORTEX ANALYST

Cortex Analyst

Question in, trusted SQL out

QUESTION

“Revenue by region last month?”

CORTEX ANALYST

Reads the semantic view, writes SQL

ANSWER

Table or chart, plus the SQL used

SEMANTIC VIEW: YOUR BUSINESS DICTIONARY

- **Tables** and keys
- **Relationships**: joins
- **Dimensions**: group by
- **Metrics**: totals

Plus synonyms (“item” = product) and verified questions with their SQL.

Create Cortex Analyst, step by step

1. Left menu: **AI & ML » Cortex Analyst**
2. **Create new » Create new Semantic View**
3. Location **CORTEX_LAB_DB.RETAIL**, name **RETAIL_SV**, a description, **Next**
4. Pick tables: ORDERS, PRODUCTS, REVIEWS_ENRICHED, **Next**
5. Select all columns, **Next**
6. Review suggested relationships and metrics, accept, **Save**
7. In the editor, add synonyms and plain descriptions
8. Ask in the chat panel, check the SQL, save good answers as **verified queries**

Hands-on: build and ask the semantic view

1. New SQL file: **08_semantic_view**, run the code

2. Test it in SQL:

```
SELECT * FROM SEMANTIC_VIEW(
 retail_sv
 DIMENSIONS products.category
 METRICS orders.total_revenue);
```

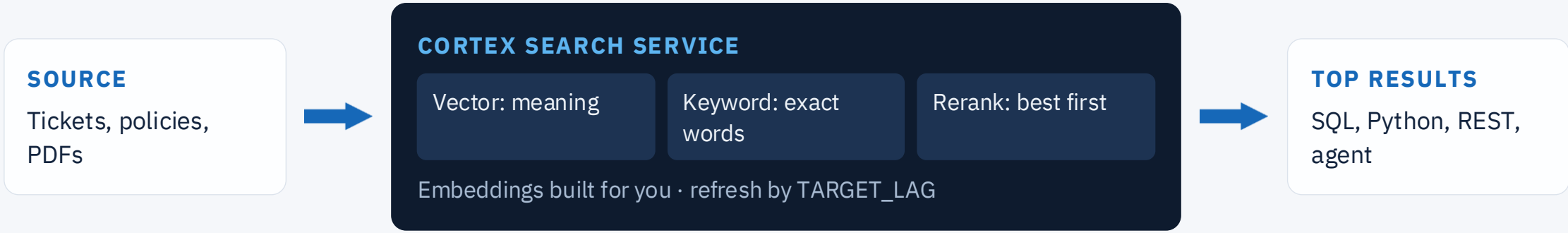
3. AI & ML » Cortex Analyst, open retail_sv, ask: “Which category sold most in Penang?”

```
CREATE OR REPLACE SEMANTIC VIEW retail_sv
 TABLES (
  orders AS cortex_lab_db.retail.orders
    PRIMARY KEY (order_id),
  products AS cortex_lab_db.retail.products
    PRIMARY KEY (product_id))
 RELATIONSHIPS (orders_to_products AS
  orders (product_id) REFERENCES products)
 DIMENSIONS (
  products.product_name AS products.product_name
    WITH SYNONYMS = ('item'),
  products.category AS products.category,
  orders.region AS orders.region,
  orders.order_date AS orders.order_date)
 METRICS (
  orders.total_revenue AS SUM(orders.revenue),
  orders.units_sold AS SUM(orders.quantity))
 COMMENT = 'Sales by product, category, region';
```

08 · CORTEX SEARCH

Cortex Search

Hybrid search, ready for RAG



RAG Question → Search finds the right passages → LLM answers **using only those passages**

Example: “router keeps dropping” finds a ticket that says “wifi disconnects every hour”.

Create Cortex Search, step by step

1. Left menu: **AI & ML » Cortex Search**, then **Create**
2. Pick a role with CORTEX_USER and the warehouse **cortex_lab_wh**
3. Database and schema **CORTEX_LAB_DB.RETAIL**, name **ticket_search**, **Next**
4. Source: **Table or view » TICKETS_ENRICHED**, **Next**
5. Columns to return: ticket_id, title, region, product_name, transcript_safe
6. Column to search: **TRANSCRIPT_SAFE**
7. Filter columns: **REGION**, **PRODUCT_NAME**
8. Target lag **1 hour**, **Create**, then test with SEARCH_PREVIEW

Syntax: create a search service

```
CREATE OR REPLACE
  CORTEX SEARCH SERVICE <name>
  ON <search_column>
  ATTRIBUTES <filter_columns>
  WAREHOUSE = <warehouse>
  TARGET_LAG = '<time>'
  [EMBEDDING_MODEL = '<model>']
  AS ( <select query> );
```

ON: the text to search

ATTRIBUTES: columns you can filter on

WAREHOUSE: builds and refreshes the index

TARGET_LAG: how stale results may be

EMBEDDING_MODEL: default is English; use **snowflake-arctic-embed-l-v2.0** for many languages

Hands-on: search support tickets

1. New SQL file: **07_search**

2. Turn on change tracking on the redacted table

3. Create the service, wait about a minute

4. Preview: words differ, meaning matches

No-code way: AI & ML » Cortex Search » Create

```
ALTER TABLE tickets_enriched SET CHANGE_TRACKING = TRUE;

CREATE OR REPLACE CORTEX SEARCH SERVICE ticket_search
  ON transcript_safe
  ATTRIBUTES region, product_name
  WAREHOUSE = cortex_lab_wh
  TARGET_LAG = '1 hour'
  AS (SELECT ticket_id, title, region,
        product_name, transcript_safe
      FROM tickets_enriched);

SELECT PARSE_JSON(SNOWFLAKE.CORTEX.SEARCH_PREVIEW(
  'cortex_lab_db.retail.ticket_search',
  '{"query": "wifi keeps disconnecting",
    "columns": ["ticket_id", "title"],
    "filter": {"@eq": {"region": "Penang"}},
    "limit": 3}'))['results'] AS results;
```

09 · AI FUNCTIONS

Cortex AI Functions

13 AI Functions in 6 families, plus 3 helpers

Generate

1 AI_COMPLETE

Sort and test

2 AI_CLASSIFY
3 AI_FILTER

Understand text

4 AI_SENTIMENT 5 AI_EXTRACT
6 AI_TRANSLATE 7 AI_REDACT

Many rows, one answer

8 AI_SUMMARIZE_AGG
9 AI_AGG

Meaning as numbers

10 AI_EMBED
11 AI_SIMILARITY

Files and media

12 AI_PARSE_DOCUMENT
13 AI_TRANSCRIBE

HELPERS TO_FILE · AI_COUNT_TOKENS · PROMPT

Rules for all: role needs **SNOWFLAKE.CORTEX_USER** · billed per token (files per page or second) · use an XS to M warehouse. Newest additions, not covered here: AI_SUMMARIZE and AI_MULTI_EMBED.

Helper functions: small tools that make AI calls safer

TO_FILE

Points an AI function at a file on a stage: image, PDF, audio.

```
TO_FILE('@stage', 'path')
```

```
AI_PARSE_DOCUMENT(
 TO_FILE('@docs',
  'policy.pdf'))
```

AI_COUNT_TOKENS

Counts tokens before you run, so you can estimate cost and stay under model limits.

```
AI_COUNT_TOKENS('<function>',
 ['<model>',] <text>)
```

```
AI_COUNT_TOKENS(
 'ai_complete',
 'claude-sonnet-4-5',
 review_text)
```

PROMPT

Builds a prompt from a template; {0}, {1} are filled from columns.

```
PROMPT('text {0} {1}',
 col0, col1)
```

```
AI_FILTER(PROMPT(
 'Asks for a refund:
 {0}', review_text))
```

AI FUNCTIONS · 1 OF 13

AI_COMPLETE: ask any LLM, from SQL

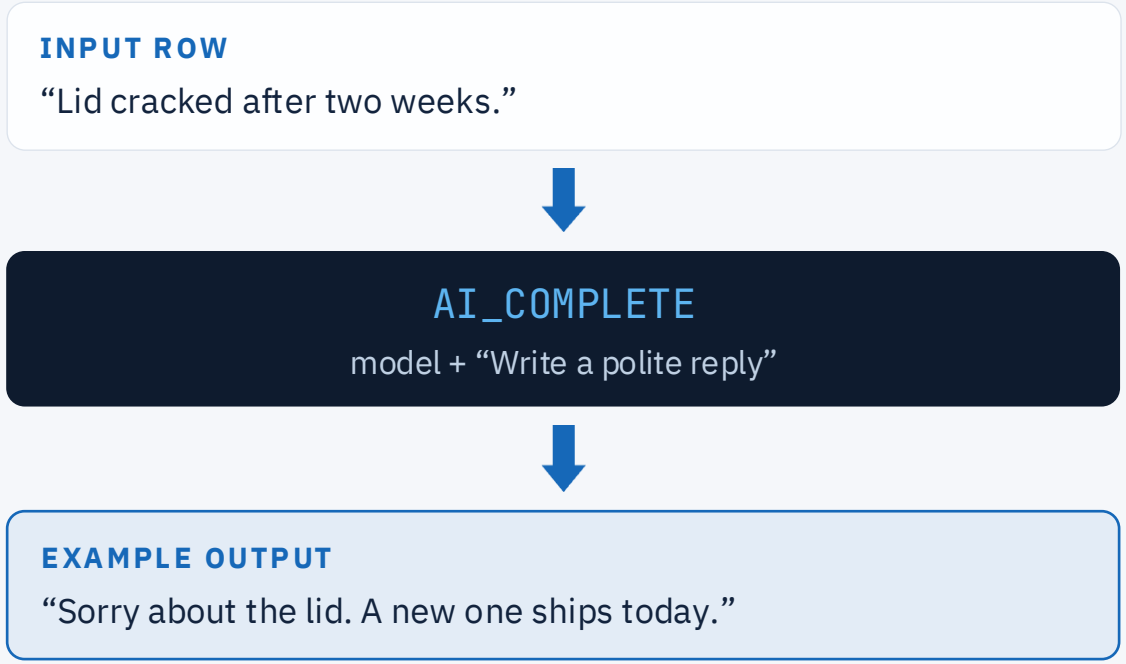
What it does

Sends your prompt to the LLM you choose and returns its answer, once for every row.

Use it for

- Draft a reply to each review
- Rewrite or summarise one long text
- Turn messy text into clean JSON
- Describe an image on a stage

Why this one: the most flexible. If a task function below fits, use it instead: simpler and cheaper.



Syntax: two forms

String completions

```
AI_COMPLETE( <model>, <prompt> [ , <model_parameters> , <response_format>, <show_details> ] )
```

Image or file input

```
AI_COMPLETE( <model>, <predicate>, <file> [ , <model_parameters> ] )
```

temperature 0 to 1, default 0

top_p 0 to 1, default 0

max_tokens default 4096

guardrails TRUE / FALSE

response_format: force JSON shape

show_details: add token usage

Green = required · Yellow = optional model settings · Files come from a stage via TO_FILE

Hands-on: AI_COMPLETE, step 1

1. Load the lab data (lab guide, step 2)

2. New SQL file: **02_ai_complete**

3. Run query 1. One question, one answer

4. Run query 2. A draft reply for each review

Model error? Turn on cross-region, or try **mistral-large2**

```
USE WAREHOUSE cortex_lab_wh;
USE SCHEMA cortex_lab_db.retail;

-- 1. One question
SELECT AI_COMPLETE('claude-sonnet-4-5',
  'Explain a data warehouse in two lines.');

-- 2. One answer per row
SELECT review_id, review_text,
  AI_COMPLETE('llama3.1-70b',
    CONCAT('Write a short, polite reply: ',
           review_text)) AS draft_reply
FROM customer_reviews
LIMIT 5;
```

Hands-on: AI_COMPLETE, step 2

5. Query 3: set temperature and max_tokens, see token usage with **show_details**

6. Query 4: force a JSON shape with **response_format**

7. Read a field: **result:urgency::STRING**

```
-- 3. Control the output
SELECT AI_COMPLETE(
  model => 'claude-sonnet-4-5',
  prompt => 'Name 3 ways to cut delivery delays.',
  model_parameters => {'temperature': 0, 'max_tokens': 200},
  show_details => TRUE) AS answer;

-- 4. Force a JSON shape
SELECT ticket_id, AI_COMPLETE(
  model => 'claude-sonnet-4-5',
  prompt => CONCAT('Read this ticket: ', transcript),
  response_format => TYPE OBJECT(
    issue STRING, urgency STRING)) AS result
FROM support_tickets LIMIT 5;
```

AI FUNCTIONS · 2 OF 13

AI_CLASSIFY: put text into your own buckets

What it does

Reads each text (or image) and picks the best label from a list **you** give it.

Use it for

- Route tickets: billing, delivery, technical
- Tag reviews by topic, then count them
- Label product photos by category

Why this one: answers are always one of your labels, so you can GROUP BY and chart them. Free text from AI_COMPLETE cannot.

INPUT

“Parcel arrived 10 days late.”



`AI_CLASSIFY`

labels: quality · delivery · price



OUTPUT

```
{"labels": ["delivery"]}
```

AI_CLASSIFY: syntax and example

```
AI_CLASSIFY(<input>,
  <list_of_categories>
  [, <config>])
```

input: text, or an image via TO_FILE

list_of_categories: 2 or more labels

config: task_description, output_mode 'multi', examples

Returns an object. Read **:labels[0]**

```
SELECT review_text,
  AI_CLASSIFY(review_text,
    ['product quality', 'delivery',
     'price', 'customer service'],
    {'task_description':
       'Main topic of a shop review'}
  ):labels[0]::STRING AS topic
FROM customer_reviews;
```

review_text (example output)	topic
Lid cracked after two weeks	product quality
Parcel arrived 10 days late	delivery

AI FUNCTIONS · 3 OF 13

AI_FILTER: a yes or no test in plain English

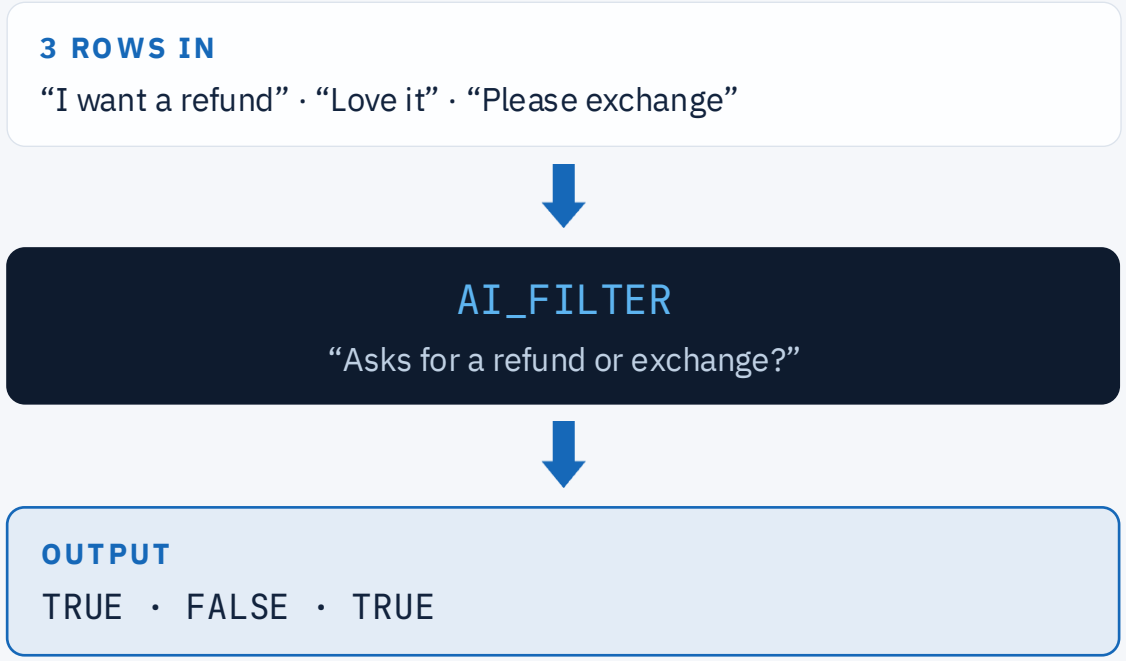
What it does

Asks your question about each row and returns **TRUE** or **FALSE**, so AI can sit inside WHERE.

Use it for

- Keep only refund requests
- Find tickets that mention a safety risk
- Flag photos that show damaged boxes

Why this one: no keyword lists to maintain. “I want my money back” and “please refund me” both pass.



AI_FILTER: syntax and example

```
AI_FILTER(<input>)

AI_FILTER(PROMPT(
  '<question> {0}', <column>))

AI_FILTER(<predicate>, <file>)
```

input: a full yes/no question

PROMPT: puts the column into {0}

Returns BOOLEAN, or NULL on error

```
-- quick test
SELECT AI_FILTER('Is Penang in Malaysia?');

-- only refund or exchange requests
SELECT review_id, review_text
FROM customer_reviews
WHERE AI_FILTER(PROMPT(
  'The customer asks for a refund
   or exchange: {0}', review_text));
```

id	review_text (example output)
1	The blender lid cracked … I want a refund.
4	… Please exchange for a bigger size.

AI_SENTIMENT: how does the customer feel?

What it does

Labels the feeling in a text: positive, negative, neutral, mixed or unknown. Overall, and per topic you name.

Use it for

- Track customer mood by product or month
- Alert a team on new negative reviews
- Compare aspects: delivery vs price

Why this one: one review can love the product and hate the delivery. Aspect labels show both.

INPUT

“Nice grip, but the parcel came 10 days late.”



`AI_SENTIMENT`

aspects: quality · delivery



EXAMPLE OUTPUT

overall: mixed · quality: positive · delivery: negative

AI_SENTIMENT: syntax and example

```
AI_SENTIMENT(<text>
  [, <categories>])
```

text: the review or message

categories: optional aspects, e.g. ['delivery', 'price']

Returns an object with a **categories** array. The first entry is **overall**

```
SELECT review_id,
  AI_SENTIMENT(review_text)
    :categories[0]:sentiment::STRING
    AS overall,
  AI_SENTIMENT(review_text,
    ['delivery', 'price', 'quality'])
    AS by_aspect
FROM customer_reviews;
```

```
{"categories": [
 {"name": "overall", "sentiment": "mixed"},
 {"name": "delivery", "sentiment": "negative"}, …]}
```

AI FUNCTIONS · 5 OF 13

AI_EXTRACT: pull facts out of text and files

What it does

Answers the questions you list, from a text or a document, and returns one field per question.

Use it for

- Order numbers from support tickets
- Total and due date from invoice PDFs
- Parties and amounts from contracts

Why this one: turns free text and files into columns you can join and report on.

INPUT TICKET

“Jug leaks from the seal. Order KDK-56010. Customer wants a full refund.”



EXAMPLE OUTPUT

```
order_no: KDK-56010 · request: refund
```

AI_EXTRACT: syntax and example

```
AI_EXTRACT(
  text => <text>
   | file => <file>,
  responseFormat =>
    {'<key>': '<question>'}
  [, scores => TRUE])
```

responseFormat: one key per fact you want

Returns {"error", "response"}. Read **:response:<key>**

```
SELECT ticket_id,
  AI_EXTRACT(
    text => transcript,
    responseFormat => {
      'order_no': 'What is the order number?',
      'request': 'Refund, exchange or fix?'}
  ):response AS facts
FROM support_tickets;
-- same idea on a PDF in a stage
SELECT AI_EXTRACT(
  file => TO_FILE('@docs', 'invoice.pdf'),
  responseFormat => {'total': 'Invoice total?'});
```

```
{"order_no": "KDK-56010", "request": "refund"}
```

AI FUNCTIONS · 6 OF 13

AI_TRANSLATE: one language for every row

What it does

Translates text between 23 languages. Leave the source empty and it detects the language.

Use it for

- Read Chinese or Japanese reviews in English
- Reply to customers in their language
- Translate first, then run other AI functions

Watch out: Malay and Tamil are not supported. Use AI_COMPLETE for Bahasa Malaysia.

INPUT

空气炸锅很好用，但是包装有点破损。



AI_TRANSLATE

from '' (detect) to 'en'



EXAMPLE OUTPUT

“The air fryer works well, but the packaging was a bit damaged.”

AI_TRANSLATE: syntax and example

```
AI_TRANSLATE(<text>,
  <source_language>,
  <target_language>)
```

Codes: 'en', 'zh', 'ja', 'ko', 'hi', 'ar', 'fr', 'de' and more, 23 in all

'' as source: detect the language, even mixed text

Returns the translated string

```
-- Chinese review to English
SELECT review_text,
  AI_TRANSLATE(review_text, '', 'en')
    AS english
FROM customer_reviews
WHERE review_id = 8;

-- Malay is not supported: use AI_COMPLETE
SELECT AI_COMPLETE('claude-sonnet-4-5',
  CONCAT('Translate to English. Reply with',
    ' the translation only: ', review_text))
  AS english
FROM customer_reviews WHERE review_id = 7;
```

AI FUNCTIONS · 7 OF 13

AI_REDACT: hide personal data

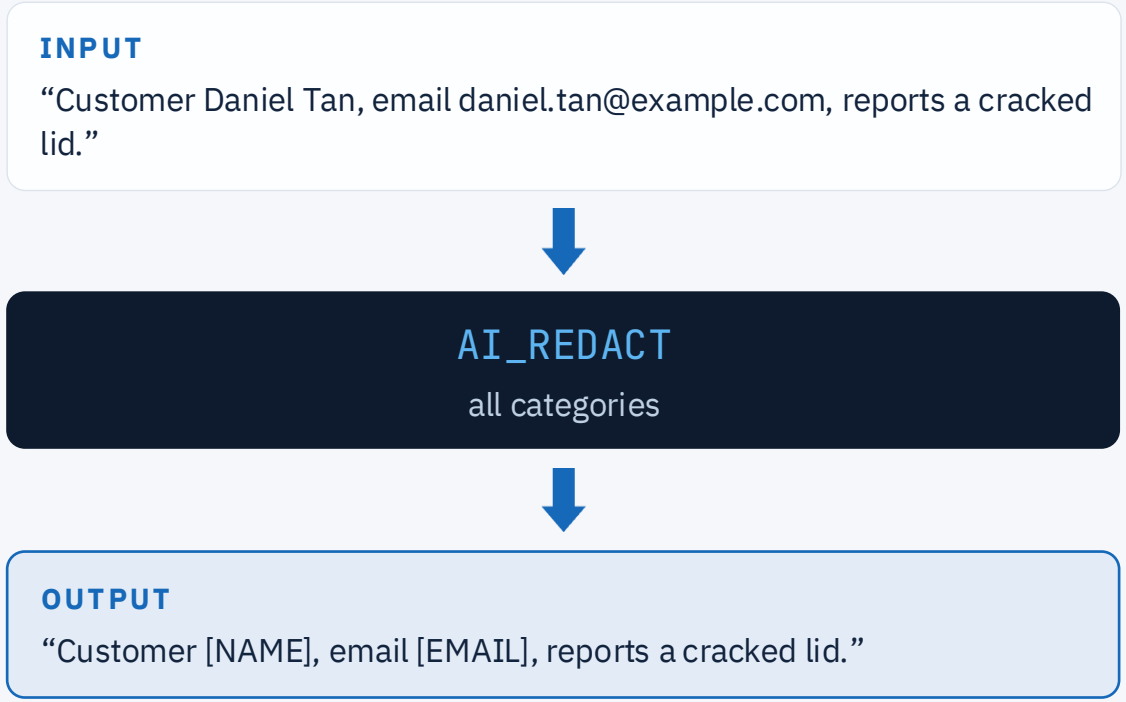
What it does

Finds personal data such as names, phones, emails, IDs and card numbers, and swaps them for tags like [NAME].

Use it for

- Share tickets with vendors or other teams
- Clean text before search or an agent sees it
- Support PDPA-friendly analytics

Why this one: 13 built-in data types, no regex to write. Detect mode shows where each item sits.



AI_REDACT: syntax and example

```
AI_REDACT(<input>
  [, <categories>]
  [, <return_error_details>]
  [, <mode>])
```

categories: NAME, EMAIL, PHONE_NUMBER, ADDRESS, NATIONAL_ID, PASSPORT, PAYMENT_CARD_DATA, DATE_OF_BIRTH and more. Empty = all

mode: 'redact' (default) or 'detect'

Returns text with tags, or in detect mode an object with **spans**

```
-- 1. hide everything it finds
SELECT ticket_id,
  AI_REDACT(transcript) AS safe_text
FROM support_tickets;

-- 2. only names and emails
SELECT AI_REDACT(
  input => transcript,
  categories => ['NAME', 'EMAIL'])
FROM support_tickets;
```

```
Customer [NAME] ([PHONE_NUMBER]) says the
wifi drops every hour … Order KDK-55821.
```

AI_SUMMARIZE_AGG: many rows, one summary

What it does

Reads every text in a group and writes one short summary, like SUM but for words.

Use it for

- One paragraph per product from all reviews
- Weekly summary of all support tickets
- Survey answers summed up per branch

Why this one: not limited by one context window, so it handles thousands of rows.

MANY ROWS

All BlendMax 900 reviews: lid cracked · noisy · burning smell · …



`AI_SUMMARIZE_AGG`

GROUP BY product



EXAMPLE OUTPUT, ONE ROW

“Buyers report cracked lids, leaks, noise and a burning smell.”

AI_SUMMARIZE_AGG: syntax and example

```
AI_SUMMARIZE_AGG(<text_column>)
```

text_column: the text to summarise

Use with GROUP BY, or alone for the whole table

Returns one summary string per group

```
SELECT p.product_name,
  COUNT(*) AS reviews,
  AI_SUMMARIZE_AGG(r.review_text) AS summary
FROM customer_reviews r
JOIN products p
  ON p.product_id = r.product_id
GROUP BY p.product_name;
```

product_name	summary (example output)
BlendMax 900	Cracked lids, leaks, noise, burning smell
AirFry Pro 5L	Works well; one damaged box

AI_AGG: your own question over many rows

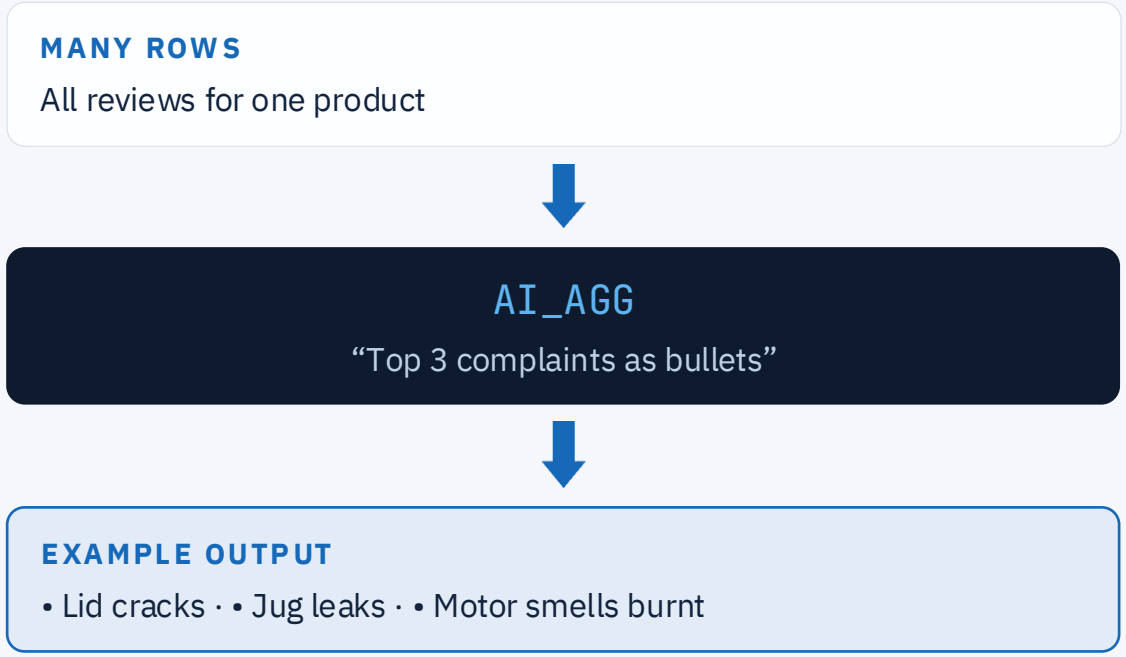
What it does

Reads every text in a group and answers **your** instruction about them, like an analyst reading a pile of notes.

Use it for

- Top 3 complaints per product
- Most requested features per region
- Risks mentioned across all tickets

Why this one: AI_SUMMARIZE_AGG only summarises. AI_AGG answers the exact question you ask.



AI_AGG: syntax and example

```
AI_AGG(<text_column>,
  '<instruction>')
```

text_column: the texts to read

instruction: what to find or list

Returns one answer string per group

```
SELECT product_name,
  AI_AGG(transcript,
    'You are a product manager. List the
     top 3 complaints as short bullets.')
  AS top_complaints
FROM support_tickets
GROUP BY product_name;
```

product_name	top_complaints (example)
BlendMax 900	Cracked lid · leaking jug · burning smell
MeshWiFi Router X3	Drops hourly · slow at night

AI FUNCTIONS · 10 OF 13

AI_EMBED: turn meaning into numbers

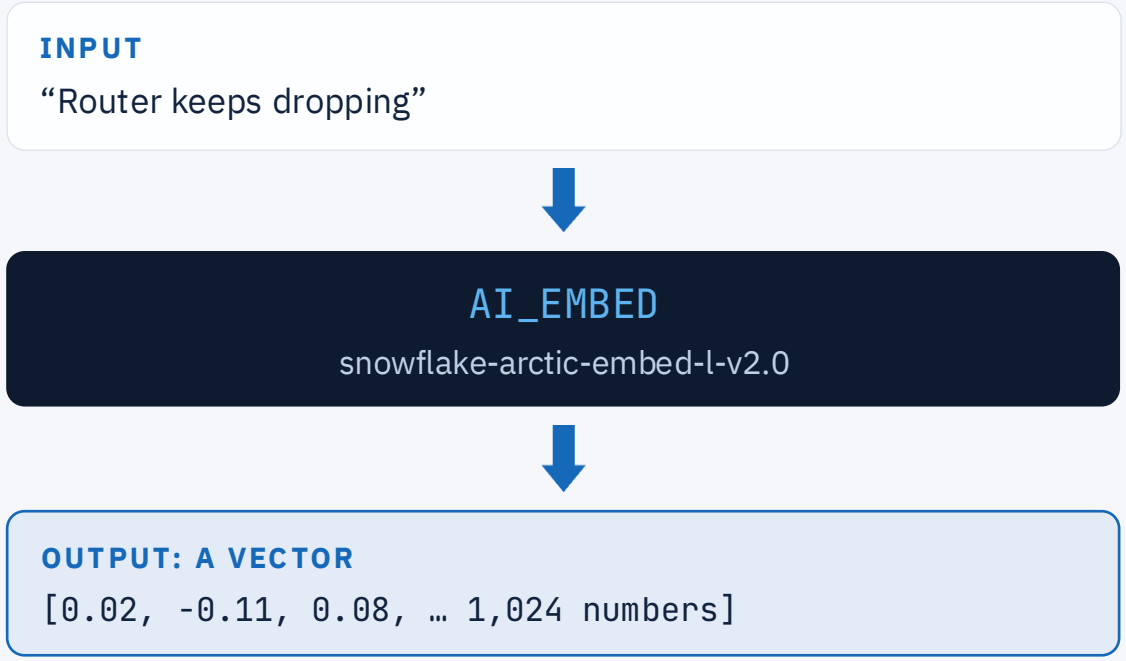
What it does

Turns a text or image into a vector: a long list of numbers. Similar meanings get close vectors.

Use it for

- Your own semantic search or RAG
- Group similar tickets together
- Find duplicate products or questions

Why this one: full control over vectors. Cortex Search does this for you when you only need search.



AI_EMBED: syntax and example

```
AI_EMBED(<model>, <input>)
```

Text models: snowflake-arctic-embed-l-v2.0 (multilingual), multilingual-e5-large, voyage-multilingual-2 and more

Image model: voyage-multimodal-3, input via TO_FILE

Returns a VECTOR

```
-- 1. store one vector per ticket
CREATE OR REPLACE TABLE ticket_vectors AS
SELECT ticket_id, AI_EMBED(
  'snowflake-arctic-embed-l-v2.0',
  transcript) AS vec
FROM support_tickets;

-- 2. closest tickets to a question
SELECT ticket_id,
  VECTOR_COSINE_SIMILARITY(vec, AI_EMBED(
    'snowflake-arctic-embed-l-v2.0',
    'wifi keeps disconnecting')) AS score
FROM ticket_vectors
ORDER BY score DESC LIMIT 3;
```

AI_SIMILARITY: how alike are two things?

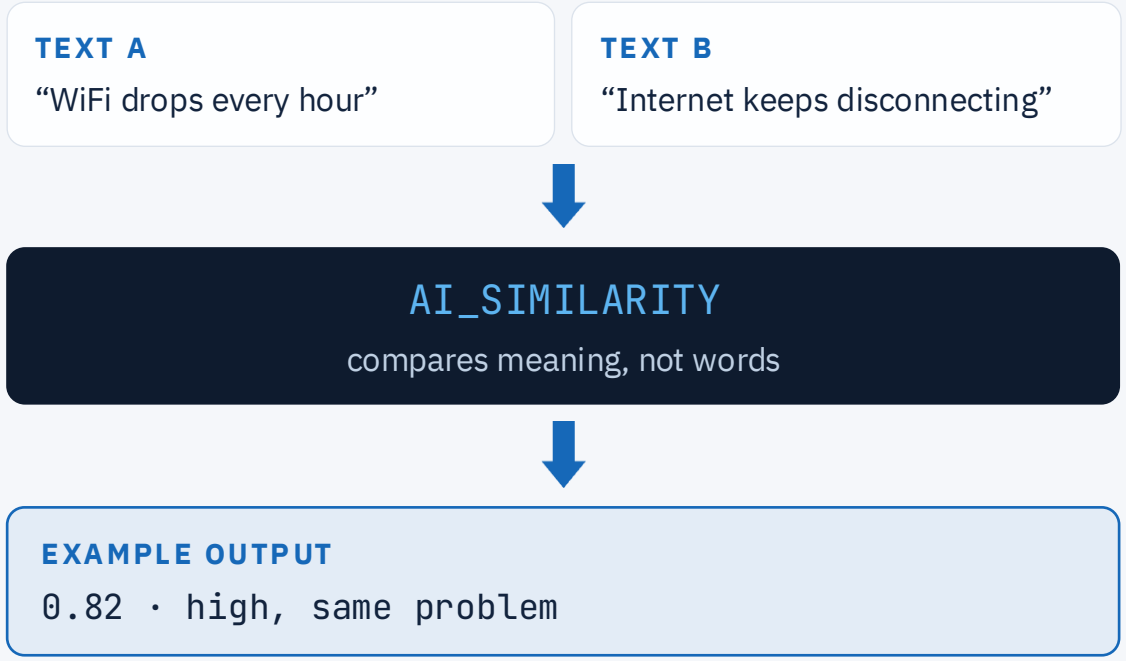
What it does

Compares two texts (or two images) by meaning and returns one score. Higher score = more alike.

Use it for

- Is this ticket a duplicate of that one?
- Match a review to the right product
- Check an answer against a reference answer

Why this one: no vectors to store. AI_EMBED is better when you compare one item against millions.



AI_SIMILARITY: syntax and example

```
AI_SIMILARITY(<input1>,
  <input2>
  [, {'model': '<embed_model>'}])
```

input1, input2: two texts, or two images via TO_FILE

model: optional embedding model

Returns a FLOAT score

```
-- possible duplicate tickets
SELECT a.ticket_id, b.ticket_id AS twin,
  AI_SIMILARITY(a.transcript,
    b.transcript) AS score
FROM support_tickets a
JOIN support_tickets b
  ON a.ticket_id < b.ticket_id
QUALIFY score > 0.75
ORDER BY score DESC;
```

ticket_id	twin	score (example)
T-1001	T-1007	0.84

AI_PARSE_DOCUMENT: PDF in, text out

What it does

Reads a file on a stage (PDF, Word, slides, image) and returns its text. LAYOUT mode keeps tables and headings as Markdown.

Use it for

- Product manuals into Cortex Search
- Invoices and contracts into tables
- Scanned forms via OCR

Why this one: the first step for any document. Other AI functions only read text.

INPUT: A STAGED FILE

```
@docs/blendmax_manual.pdf
```



AI_PARSE_DOCUMENT

mode: OCR or LAYOUT



OUTPUT: JSON OBJECT

```
{"content": "# Safety\n| Part | … |",
 "metadata": {"pageCount": 12}}
```

AI_PARSE_DOCUMENT: syntax and example

```
AI_PARSE_DOCUMENT(
  TO_FILE('@stage', 'file'),
  {'mode': 'OCR' | 'LAYOUT',
   'page_split': TRUE | FALSE})
```

TO_FILE: points at the staged file

mode: OCR (text) or LAYOUT (tables)

Returns OBJECT: content, metadata

```
CREATE OR REPLACE TABLE manual_text AS
SELECT relative_path,
  AI_PARSE_DOCUMENT(
    TO_FILE('@docs', relative_path),
    {'mode': 'LAYOUT'}
  ):content::STRING AS body
FROM DIRECTORY(@docs)
WHERE relative_path ILIKE '%.pdf';
```

Next: chunk body with SPLIT_TEXT_RECURSIVE_CHARACTER, then index it with Cortex Search.

AI_TRANSCRIBE: audio and video into text

What it does

Listens to a staged audio or video file and writes down what was said. Can add timestamps and tell speakers apart.

Use it for

- Call-centre recordings into tickets
- Meeting notes and minutes
- Searching inside training videos

Why this one: turns sound into text, so every other AI function can then work on it.

INPUT: A STAGED FILE

```
@calls/call_1001.mp3
```



AI_TRANSCRIBE

optional: word or speaker timestamps



OUTPUT: JSON OBJECT

```
{"audio_duration": 94.2,
 "text": "Hi, my router keeps…"}
```

AI_TRANSCRIBE: syntax and example

```
AI_TRANSCRIBE(
  TO_FILE('@stage', 'file')
  [, {'timestamp_granularity':
    'word' | 'speaker'}])
```

TO_FILE: the audio or video file

timestamp_granularity: optional

Returns OBJECT: text, audio_duration (and segments)

```
CREATE OR REPLACE TABLE call_text AS
SELECT relative_path,
  AI_TRANSCRIBE(
    TO_FILE('@calls', relative_path)
  ):text::STRING AS transcript
FROM DIRECTORY(@calls)
WHERE relative_path ILIKE '%.mp3';
```

relative_path	transcript (example)
call_1001.mp3	Hi, my router keeps dropping…

Watch the cost

Warehouse: XS to M. Bigger is not faster for AI

Test: LIMIT 5 first

Model: smallest that does the job

Prefer: task functions over long AI_COMPLETE prompts

Trial: about 10 AI credits a day

```
-- 1. Estimate tokens before you run
SELECT SUM(AI_COUNT_TOKENS('ai_complete',
  'claude-sonnet-4-5', review_text)) AS tokens
FROM customer_reviews;

-- 2. Daily AI credits (can lag a few hours)
SELECT usage_date, credits_used
FROM SNOWFLAKE.ACCOUNT_USAGE.METERING_DAILY_HISTORY
WHERE service_type = 'AI_SERVICES'
ORDER BY usage_date DESC;

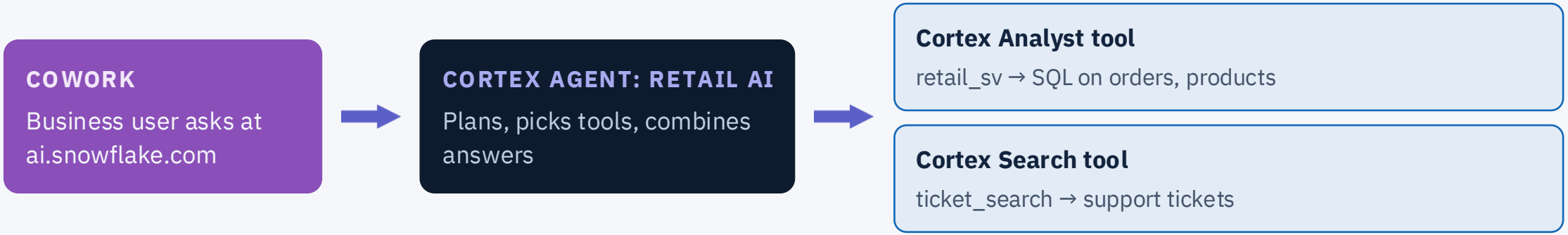
-- 3. Credits per function and model
SELECT * FROM SNOWFLAKE.ACCOUNT_USAGE
  .CORTEX_FUNCTIONS_USAGE_HISTORY LIMIT 20;
```

10 · TOGETHER AT SCALE

Cortex Agents and CoWork

AI architecture: putting the pieces together

AI architecture with CoWork



EXAMPLE

“Why did blender sales drop in Penang?” → Analyst finds the drop, Search finds the tickets about broken lids, the agent explains both with a chart.

Every query runs with the user’s role and default warehouse

RBAC · masking · per-user quota

Hands-on: publish the agent to CoWork

1. Run CREATE AGENT (previous slide)

2. Run the code on the right

3. Open **ai.snowflake.com**, pick **Retail AI**

4. Ask: “Top 3 complaints about blenders?” and “Revenue by category this year”

```
-- once per account (skip if SHOW finds it)
SHOW SNOWFLAKE INTELLIGENCES;
CREATE SNOWFLAKE INTELLIGENCE
  SNOWFLAKE_INTELLIGENCE_OBJECT_DEFAULT;

-- show the agent in CoWork
ALTER SNOWFLAKE INTELLIGENCE
  SNOWFLAKE_INTELLIGENCE_OBJECT_DEFAULT
  ADD AGENT cortex_lab_db.retail.retail_ai;

-- who may use it
GRANT USAGE ON SNOWFLAKE INTELLIGENCE
  SNOWFLAKE_INTELLIGENCE_OBJECT_DEFAULT TO ROLE PUBLIC;
GRANT USAGE ON AGENT cortex_lab_db.retail.retail_ai
  TO ROLE cortex_lab_role;
```

10 · TOGETHER AT SCALE

Per-User Quota for AI Services

A spending limit for each person

USER SPENDS

AI Functions, CoCo, Agents, CoWork

→

QUOTA OBJECT

Daily, weekly or monthly limit per user

→

80%: email warning

100%: AI requests blocked

New cycle: unblocked

Who

All users, or a group chosen by tag

When

Checked within minutes; resets at 00:00 UTC

Watch out

One limit per quota; big queries can overshoot a little

Scale · Per-user quotas, generally available August 2026 · Different limits for power users: a second quota scoped by tag

Hands-on: a daily AI quota

1. New SQL file: **10_quota**

2. Run all. 2 credits a day per user

3. Check: **GET_CONFIG()**

No-code way: Admin » Cost management » Budgets » + Budget » Quota

```
USE ROLE ACCOUNTADMIN;
CREATE SCHEMA IF NOT EXISTS cortex_lab_db.cost;
USE SCHEMA cortex_lab_db.cost;
CREATE SNOWFLAKE.CORE.QUOTA lab_ai_quota();

-- what to count
CALL lab_ai_quota!ADD_SHARED_RESOURCE('AI FUNCTION');
CALL lab_ai_quota!ADD_SHARED_RESOURCE('CORTEX CODE');
CALL lab_ai_quota!ADD_SHARED_RESOURCE('CORTEX AGENT');
CALL lab_ai_quota!ADD_SHARED_RESOURCE('SNOWFLAKE INTELLIGENCE');

-- limit, warning, block
CALL lab_ai_quota!SET_PER_USER_LIMIT(2, 'DAILY');
CALL lab_ai_quota!ADD_NOTIFICATION_THRESHOLD(80, 'ACTUAL', TRUE, 'DAILY');
CALL lab_ai_quota!SET_BLOCK_ENFORCEMENT_ENABLED(TRUE, TRUE);
CALL lab_ai_quota!GET_CONFIG();
```

Scale · SNOWFLAKE INTELLIGENCE = the CoWork domain · CORTEX CODE = the CoCo domain

10 · TOGETHER AT SCALE

CoCo

Snowflake’s AI coding agent, formerly Cortex Code

CoCo builds. CoWork answers.

	CoCo	CoWork
For	Engineers, analysts, admins	Business users
Where	Snowsight, CLI, Desktop app	ai.snowflake.com, Slack, Excel
Does	Writes and runs SQL, Python, dbt, apps	Answers with agents, charts, documents
Quota domain	CORTEX CODE	SNOWFLAKE INTELLIGENCE

Both: Claude models, your roles and data rules, cross-region inference turned on.

What you can now build

1. AI on every row with AI Functions
2. Meaning-based search over tickets
3. English questions to SQL with Analyst
4. An agent that uses both tools
5. CoWork for business users
6. Per-user quotas and MCP access

Next: open the **hands-on lab guide** and build the Feedback Intelligence app end to end.