



Shifting Left and Moving Forward

Modern Data Development with MotherDuck and Dagster

January 14, 2025

Table of contents

01

Introduction

02

Bluesky end-to-end project

03

Power of MotherDuck

04

Shifting left in data engineering

05

Discussion on modern data workflows

06

Q & A

Speakers



Colton Padden

Developer Advocate



Alex Noonan

Developer Advocate



Jacob Matson

Developer Advocate

End to End Dagster Example



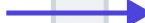
Ingestion



Bluesky



R2 Bucket



Transformation



dbt



Motherduck
Warehouse



Reporting



PowerBI

https://github.com/dagster-io/dagster/tree/master/examples/project_atproto_dashboard

Ingestion

Bluesky SDK



Bluesky

Docs

Blog

Showcase

GitHub



Search



Get Started

Tutorials

Starter Templates

Advanced Guides

HTTP Reference

Support



> Get Started

Get Started

Make your first post to the Bluesky app via the API in under 5 minutes.

Install the SDK

Choose the SDK you want to work with. Below, we use TypeScript and Python. You can follow the instructions for the community-maintained [atproto.dart package here](#).

TypeScript

CURL

Python

Install [atproto](#) using your preferred package manager.

```
pip install atproto
```

Install the SDK

Create a session

Create a post

Next Steps

Create a session

Ingestion

Dagster Resource

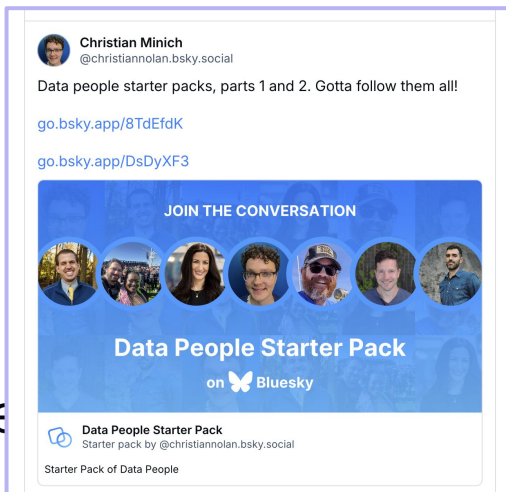
- Created an ATProtoResource
- Wraps the `atproto` SDK
- Creates a shared session across resource instantiations

```
1 class ATProtoResource(dg.ConfigurableResource):
2     login: str
3     password: str
4     session_cache_path: str = "atproto-session.txt"
5
6     def _login(self, client):
7         """Create a re-usable session to be used across resource instances.
8
9         We are rate limited to 30 requests / 5 minutes or 300 requests day.
10        """
11         if os.path.exists(self.session_cache_path):
12             with open(self.session_cache_path) as f:
13                 session_string = f.read()
14                 client.login(session_string=session_string)
15         else:
16             client.login(login=self.login, password=self.password)
17             session_string = client.export_session_string()
18             with open(self.session_cache_path, "w") as f:
19                 f.write(session_string)
20
21     def get_client(
22         self,
23     ) -> Client:
24         client = Client()
25         self._login(client)
26         return client
```

Ingestion

Starter pack snapshots

- We create a snapshot of the “Data People Starter Pack”
- These members are used as dynamic partitions for user feed snapshots



```
1 @dg.asset(  
2     partitions_def=dg.StaticPartitionsDefinition(  
3         partition_keys=[  
4             "at://did:plc:lc5jzrr425fyah724df3z5ik/app.bsky.graph.starterpack/3l7cddlz5ja24",  
5         ]  
6     ),  
7     automation_condition=dg.AutomationCondition.on_cron("0 0 * * *"),  
8     kinds={"python"},  
9     group_name="ingestion",  
10 )  
11 def starter_pack_snapshot(  
12     context: dg.AssetExecutionContext,  
13     atproto_resource: ATProtoResource,  
14     s3_resource: S3Resource,  
15 ) -> dg.MaterializeResult:  
16     # ...  
17  
18     list_items = get_all_starter_pack_members(atproto_client, starter_pack_uri)  
19  
20     # ...  
21  
22     context.instance.add_dynamic_partitions(  
23         partitions_def_name="atproto_did_dynamic_partition",  
24         partition_keys=[list_item_view.subject.did for list_item_view in list_items],  
25     )  
26  
27     return dg.MaterializeResult(  
28         metadata={  
29             "len_members": len(list_items),  
30             "s3_object_key": object_key,  
31         }  
32     )  
33 )
```


Ingestion

User feed snapshots

- Dynamic partitions
 - Partition mapping
- Automation conditions
- Files land in CloudFlare R2
 - S3 compatible cloud storage that's way easier to configure

dagster-demo

Public URL Access: Not allowed | Domains: Not allowed | Bucket Size: 1.63 GB | Class A Operations: 0 | Class B Operations: 0

Objects Metrics Settings

Search objects by prefix:

dagster-demo / atproto_actor_feed_snapshot / 2024-12-18 / 11/

Objects	Type
04/	Direct
05/	Direct
06/	Direct
07/	Direct
08/	Direct
09/	Direct
10/	Direct
11/	Direct
12/	Direct
13/	Direct
14/	Direct
15/	Direct
16/	Direct
17/	Direct
18/	Direct
19/	Direct
20/	Direct
21/	Direct
22/	Direct
23/	Direct
24/	Direct
25/	Direct
26/	Direct
27/	Direct
28/	Direct

```
1 @dg.asset(
2     partitions_def=atproto_did_dynamic_partition,
3     deps=[dg.AssetDep(
4         starter_pack_snapshot,
5         partition_mapping=dg.AllPartitionMapping()
6     )],
7     automation_condition=dg.AutomationCondition.eager(),
8     kinds={"python"},
9     group_name="ingestion",
10    op_tags={"dagster/concurrency_key": "ingestion"},
11 )
12 def actor_feed_snapshot(
13     context: dg.AssetExecutionContext,
14     atproto_resource: ATProtoResource,
15     s3_resource: S3Resource,
16 ) -> dg.MaterializeResult:
17     """Snapshot of full user feed written to S3 storage."""
18     client = atproto_resource.get_client()
19     actor_did = context.partition_key
20
21     items = get_all_feed_items(client, actor_did)
22
23     # ...
24
25     s3_resource.get_client().put_object(
26         Body=_bytes, Bucket=AWS_BUCKET_NAME, Key=object_key
27     )
28
29     return dg.MaterializeResult(
30         metadata={
31             "len_feed_items": len(items),
32             "s3_object_key": object_key,
33         }
34     )
```

Ingestion

Resiliency (Tenacity)

- Rate limiting
- Back-off retries w/ **Tenacity**
- Dagster concurrency limits

```
1 # dagster.yaml
2
3 run_coordinator:
4   module: dagster.core.run_coordinator
5   class: QueuedRunCoordinator
6
7 concurrency:
8   default_op_concurrency_limit: 1
```

```
1 def get_all_feed_items(client: Client, actor: str) -> list["models.AppBskyFeedDefs.FeedViewPost"]:
2     """Retrieves all author feed items for a given `actor`."""
3     import math
4
5     import tenacity
6
7     @tenacity.retry(
8         stop=tenacity.stop_after_attempt(5),
9         wait=tenacity.wait_fixed(math.ceil(60 * 2.5)),
10    )
11    def _get_feed_with_retries(client: Client, actor: str, cursor: Optional[str]):
12        return client.get_author_feed(actor=actor, cursor=cursor, limit=100)
13
14    feed = []
15    cursor = None
16    while True:
17        data = _get_feed_with_retries(client, actor, cursor)
18        feed.extend(data.feed)
19        cursor = data.cursor
20        if not cursor:
21            break
22
23    return feed
```

Connecting dbt to MotherDuck

Reading right from r2

- S3 connection in dbt for r2
- One line to switch from local development and cloud production.

sources.yml

```
1 bluesky:
2   target: prod
3   outputs:
4     dev:
5       type: duckdb
6       schema: bluesky_dev
7       path: "local.duckdb"
8       threads: 1
9     extensions:
10      - httpfs
11     settings:
12       s3_region: "auto"
13       s3_access_key_id: "{{ env_var('AWS_ACCESS_KEY_ID') }}"
14       s3_secret_access_key: "{{ env_var('AWS_SECRET_ACCESS_KEY') }}"
15       s3_endpoint: "{{ env_var('AWS_ENDPOINT_URL') | replace('https://', '') }}"
16   prod:
17     type: duckdb
18     schema: bluesky
19     path: "md:prod_bluesky"
20     threads: 1
21     extensions:
22      - httpfs
23     settings:
24       s3_region: "auto"
25       s3_access_key_id: "{{ env_var('AWS_ACCESS_KEY_ID') }}"
26       s3_secret_access_key: "{{ env_var('AWS_SECRET_ACCESS_KEY') }}"
27       s3_endpoint: "{{ env_var('AWS_ENDPOINT_URL') | replace('https://', '') }}"
28
```

Transformation

Reading right from r2 `sources.yml`

- Upstream Dagster asset mapping
- Blob select Json files

```
1 version: 2
2
3 sources:
4   - name: r2_bucket
5     meta:
6       external_location: "read_ndjson_objects('r2://dagster-demo/atproto_{name}/**/*.json',
7         filename=true)"
8     tables:
9       - name: actor_feed_snapshot
10         description: "external r2 bucket with json files of actor feeds"
11       - name: starter_pack_snapshot
12         description: "external r2 bucket with json files for feed snapshots"
13
```

`stg_profiles.sql`

```
1 select * from {{ source('r2_bucket', 'starter_pack_snapshot') }}
```

PowerBI

- Business intelligence in your asset graph!
- Clearly map upstream dependencies
- Superior dashboard debugging experience



End-to-End Lineage and Control

Ingestion & Transformation

Integrate with best in class tools and build custom logic to give stakeholders the data they need when they need it.

Testing & BI

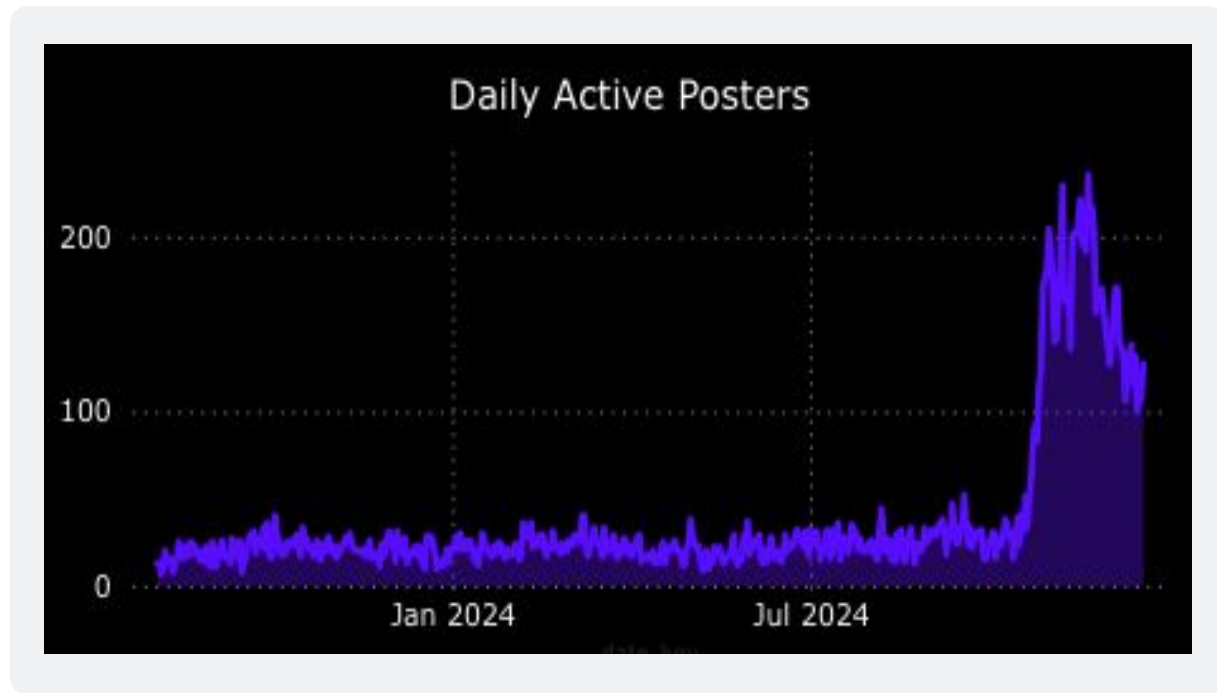
Catch issues before they show up in dashboards while give business intelligence users better insight to data assets and their lineage.

System of Record

The orchestrator touches all systems and processes in your data platforms. Making it the natural tool to build your system of record.

Flash in the Pan?

- Starterpacks
- Keeping the Dream Alive
- Outbound Links!



Fast Feedback Loops: Dagster 🤝 MotherDuck

What is DuckDB?

DuckDB is an open-source, in-process, SQL, OLAP query engine for insanely fast analytics

◆ Efficient by Design:

Automatic multi-threading, morsel-driven parallelism, and WebAssembly (Wasm) support

◆ Unparalleled Ergonomics:

Advanced SQL features, cross-format queries, and geospatial support

◆ Dynamic File Format Compatibility:

Query CSV, Parquet, JSON, Iceberg, Delta, dataframes, and more — no data movement needed



Input table: 1,000,000,000 rows x 7 columns (55 GB)

DuckDB*	1.1.0	2024-09-26	110s
Polars	1.8.2	2024-09-30	134s
InMemData.jl	0.7.21	2024-09-30 CSV import	Segfault
DataFrames.jl	1.6.1	2024-09-30 CSV import	Segfault
data.table	1.16.99	2024-09-17	timeout
dplyr	1.1.4	2024-09-17	out of memory
pandas	2.2.2	2024-09-17	out of memory
(py)datatable	1.2.0a0	2024-09-17	out of memory
spark	3.5.2	2024-09-17	timeout
dask	2024.9.0	2024-09-17	out of memory
R-arrow	17.0.0.1	2024-09-17	out of memory
Datafusion	41.0.0	2024-09-17	undefined exception
ClickHouse	24.8.4.13	2024-09-17	undefined exception
collapse	2.0.16	2024-09-17	undefined exception
Modin		see README	pending

It's taking flight



The MotherDuck Architecture

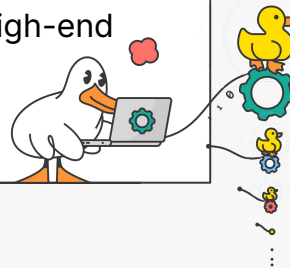
Effortless and Serverless

- Pay only for what you need
- Zero resource or system management
- Sub-100ms cold start response
- No warm up, wind down, or IO costs



Scalably Single Node

- Scale out with dedicated execution environments per user or tenant token
- No more resource contention or noisy neighbor problems
- Efficient network costs
- Optimized for high-end hardware



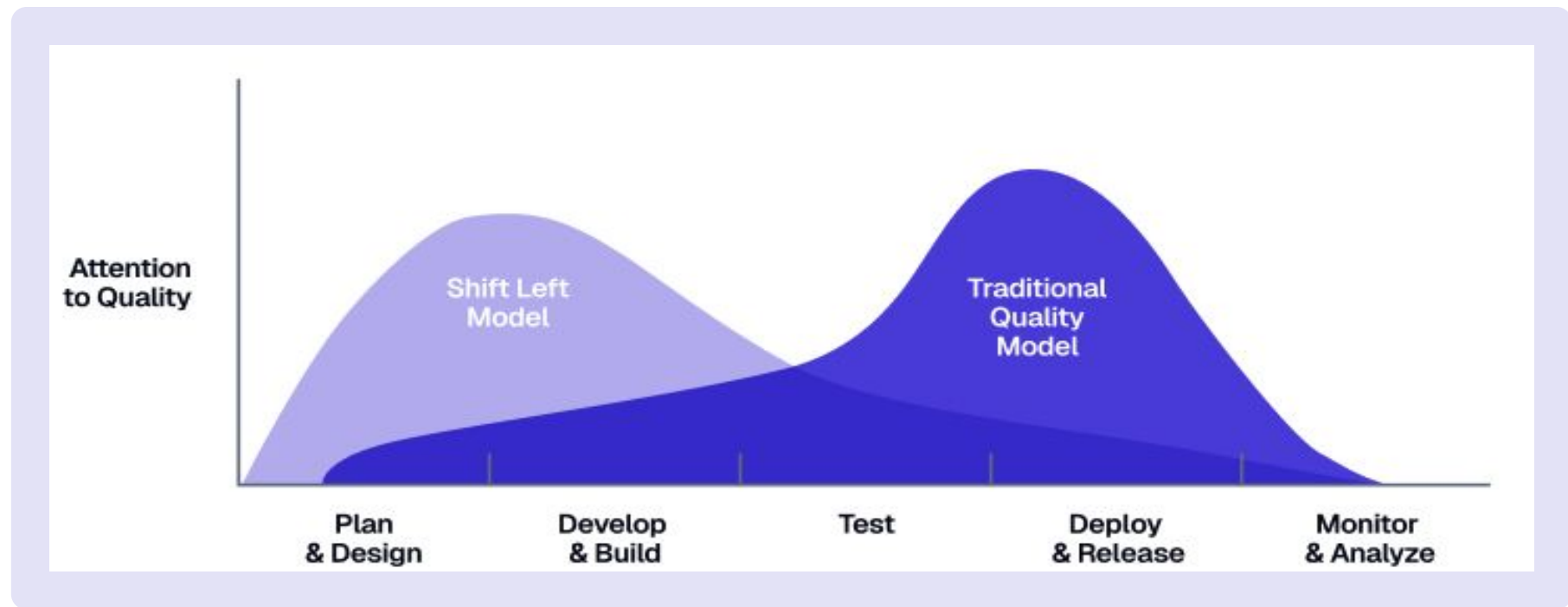
Dual Execution Query Engine

- Run sub-10ms ad-hoc queries locally
- A local environment that exactly matches the cloud
- Efficiency through minimized data movement
- Cache/buffer data locally
- Adaptive query planning executes in the best place *automagically*

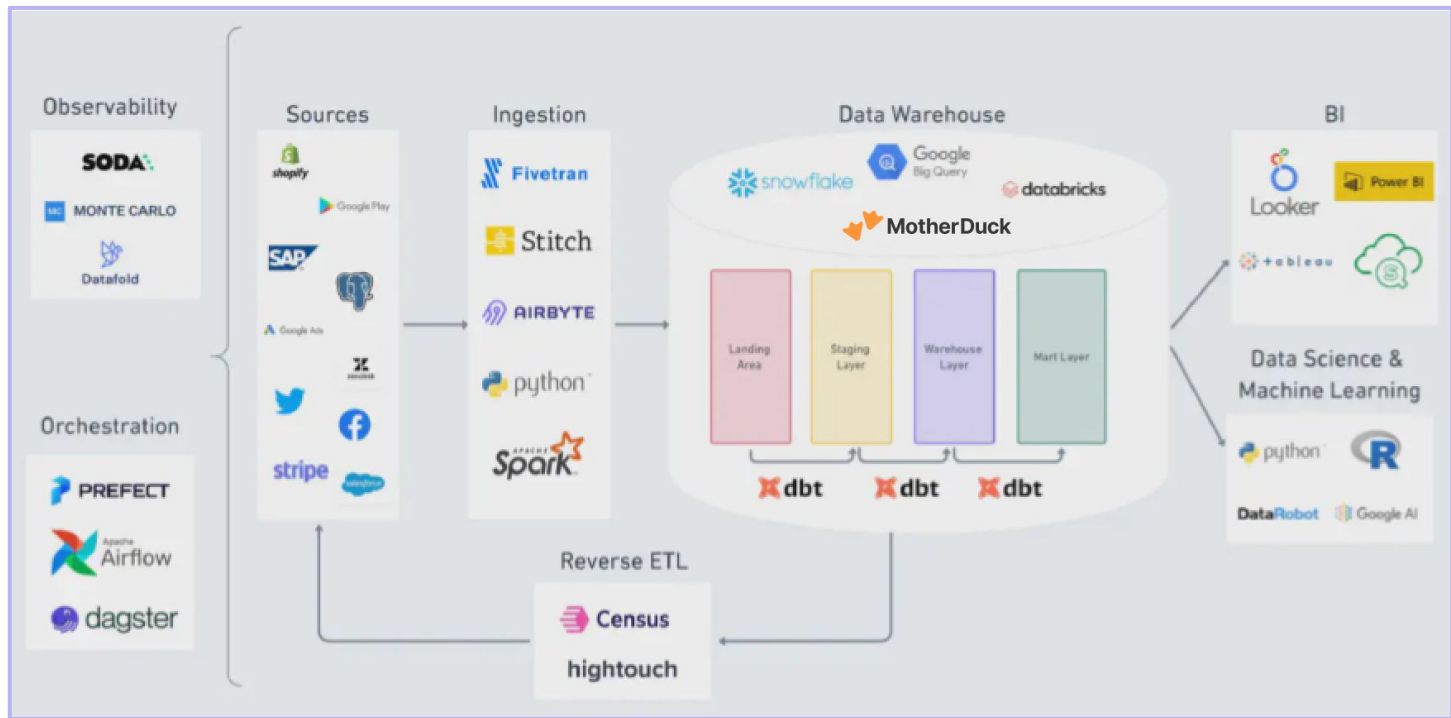


Shifting Left in Data Engineering

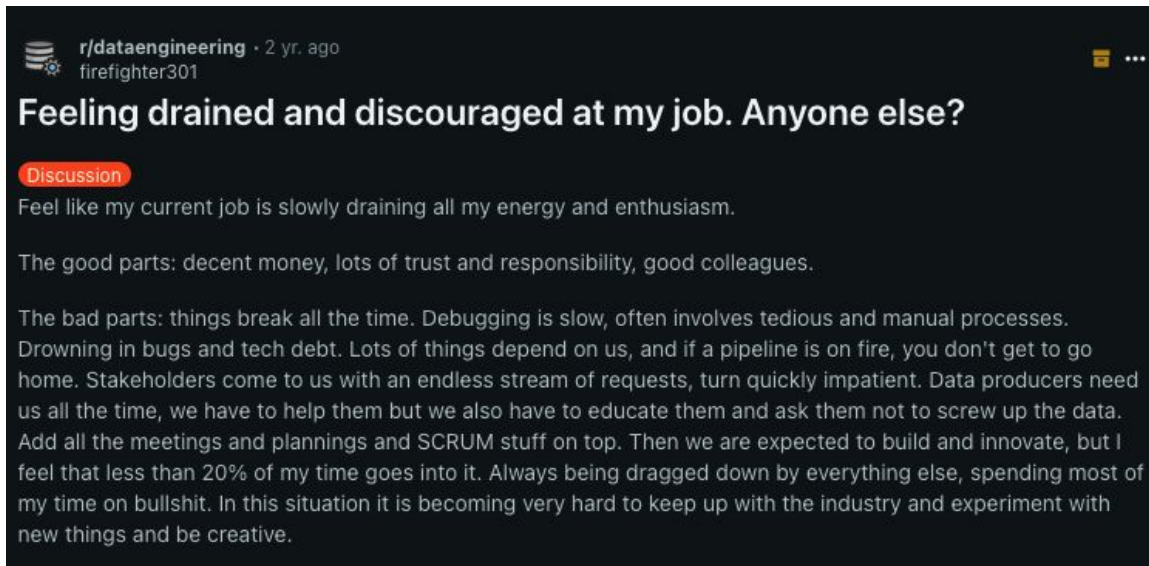
Shift Left?



Platform complexity



Drowning in Reactive Work



The Path Forward

Environment Configuration



Local Development

Making your local as close to prod as possible. No more push and pray



CI/CD

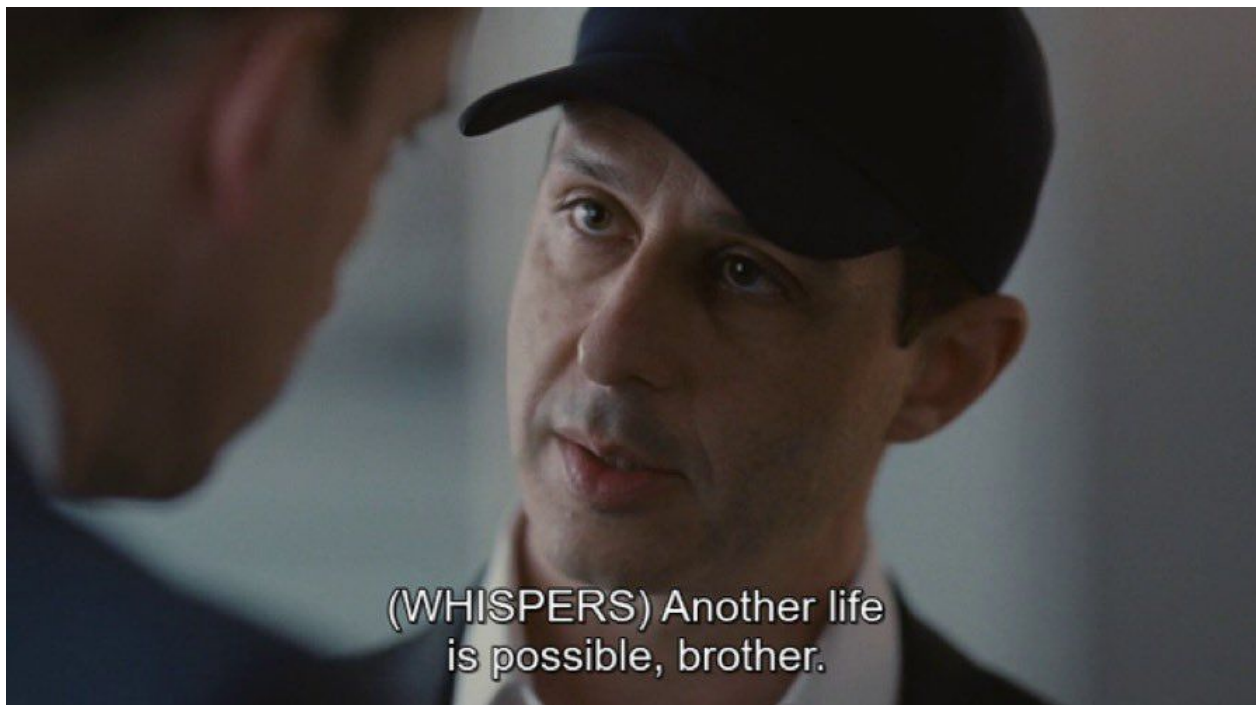
Catching things before they go to prod. Branch deployments using live data in a staging environment.



Access and Permissions

Credentials for key systems and tools. Governance without introducing excess friction.

Shift Left with Dagster



(WHISPERS) Another life
is possible, brother.

Asset Checks

Dagster OSS

- Catch data quality issues and schema changes early using Python-based logic.
- Freshness checks provide a way to identify data assets that are overdue for an update.
- dbt tests come through as asset checks

The screenshot shows the 'Overview' page for Asset Checks in Dagster. The top navigation bar includes links for Overview, Runs, Catalog, Jobs, Automation, Insights, and Deployment. The main content area is titled 'Overview' and features a 'Reload definitions' button. Below this, there are tabs for Timeline, Asset health (selected), Auto-materialize, Resources, and Backfills. A filter section allows users to select 'All assets', apply a 'Filter', and enter 'Filter asset keys...'. A 'Group by' dropdown is set to 'Asset Group'. The summary section displays four metrics: 'Execution failures 23 assets' (with a warning icon), 'Check failure errors 0 assets' (with an error icon), 'Check failure warning 1 asset' (with a warning icon and a highlighted orange bar), and 'Executing 22 assets' (with a refresh icon). Below the summary is a table with columns for Asset, Code location, Group, Checks, and Actions. The table lists one asset: 'self_serve_customer_attribution' located at 'purina' within the 'billing_platform' group, with a check named 'self_se...tions' showing a warning. A 'View asset' button is available for this entry.

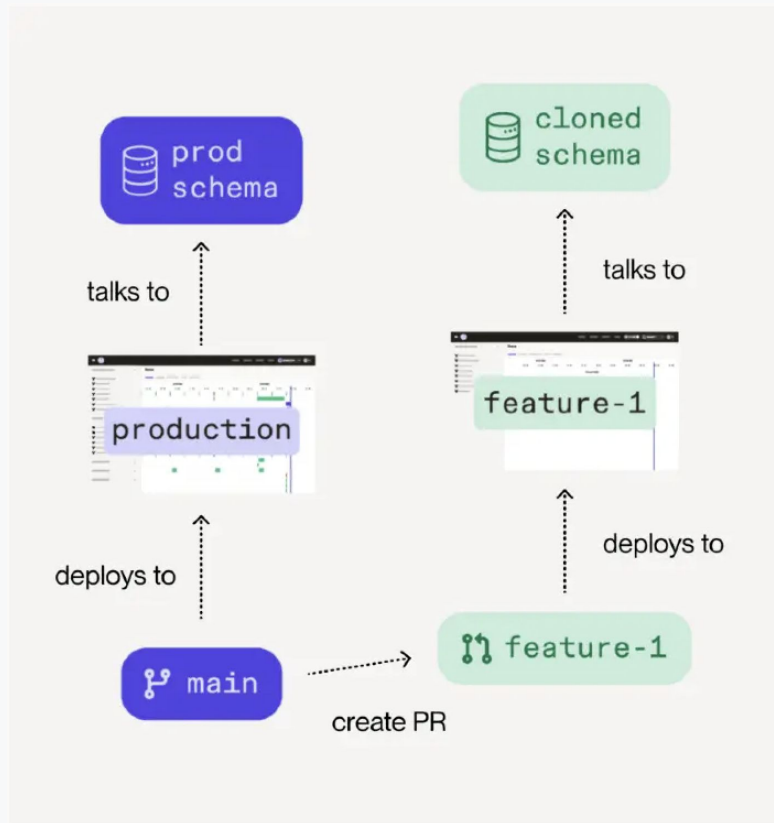
Asset	Code location	Group	Checks	Actions
self_serve_customer_attribution	purina	billing_platform	self_se...tions	View asset

Branch Deployments

Dagster+

Don't Test in Production

Every git branch push can automatically create a unique Dagster deployment for a real-time staging preview



Modern Data Workflows

The brave new world of AI-assisted development



AI Chat Apps

There's still a ton of alpha in using ChatGPT & Claude's chat interface.



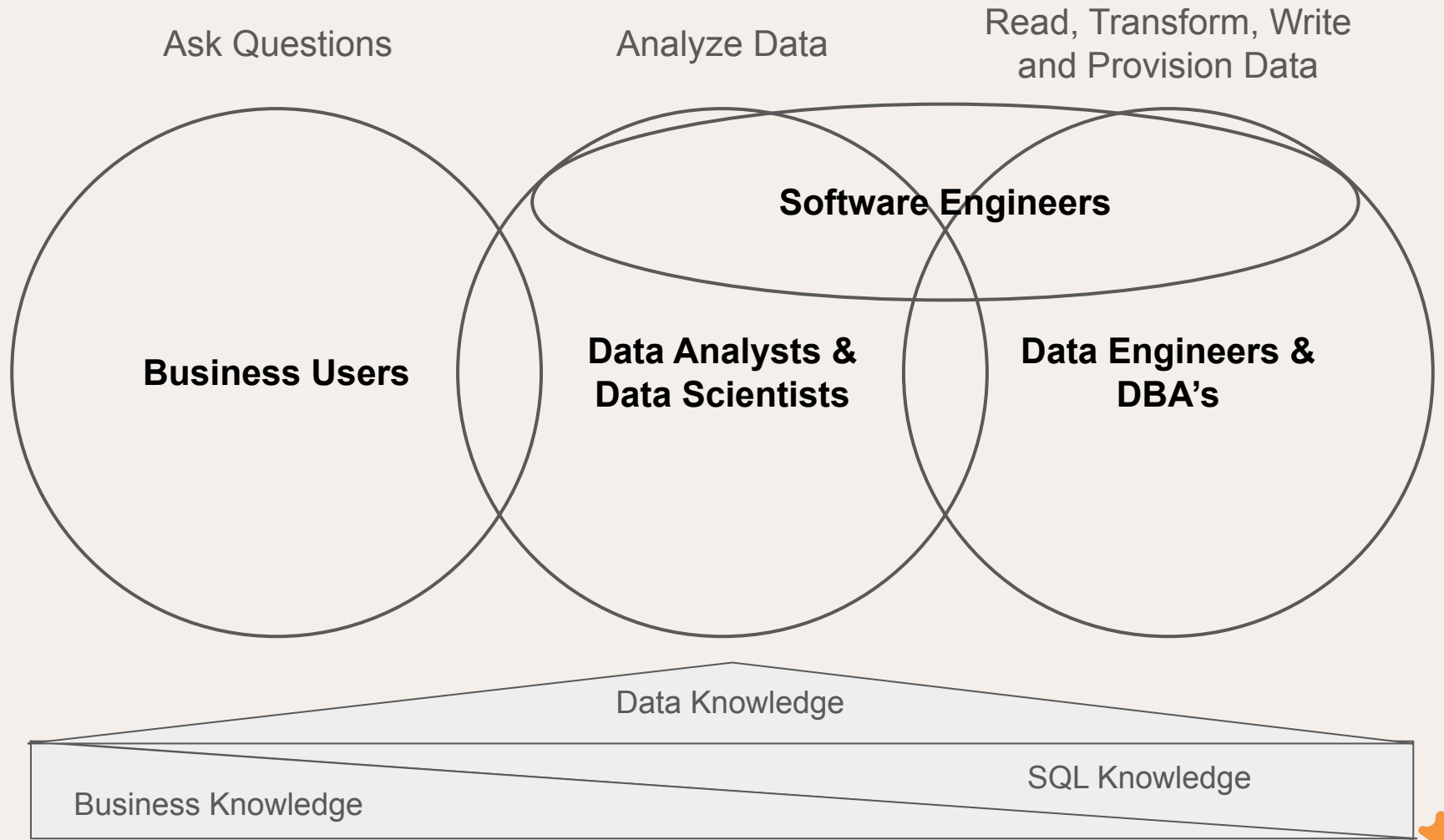
RAGs

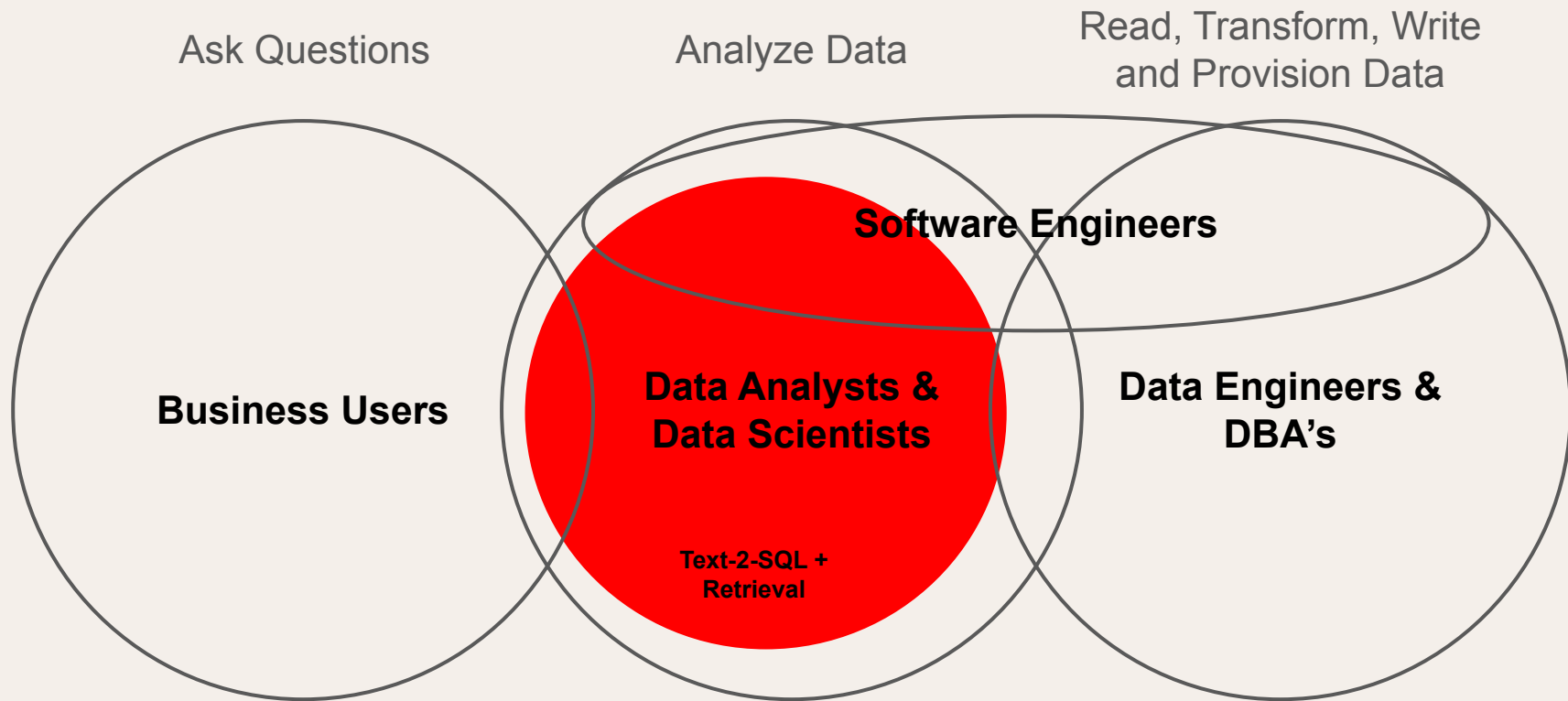
Use embedded search & relevant context to focus models around your specific use case.



AI based IDEs

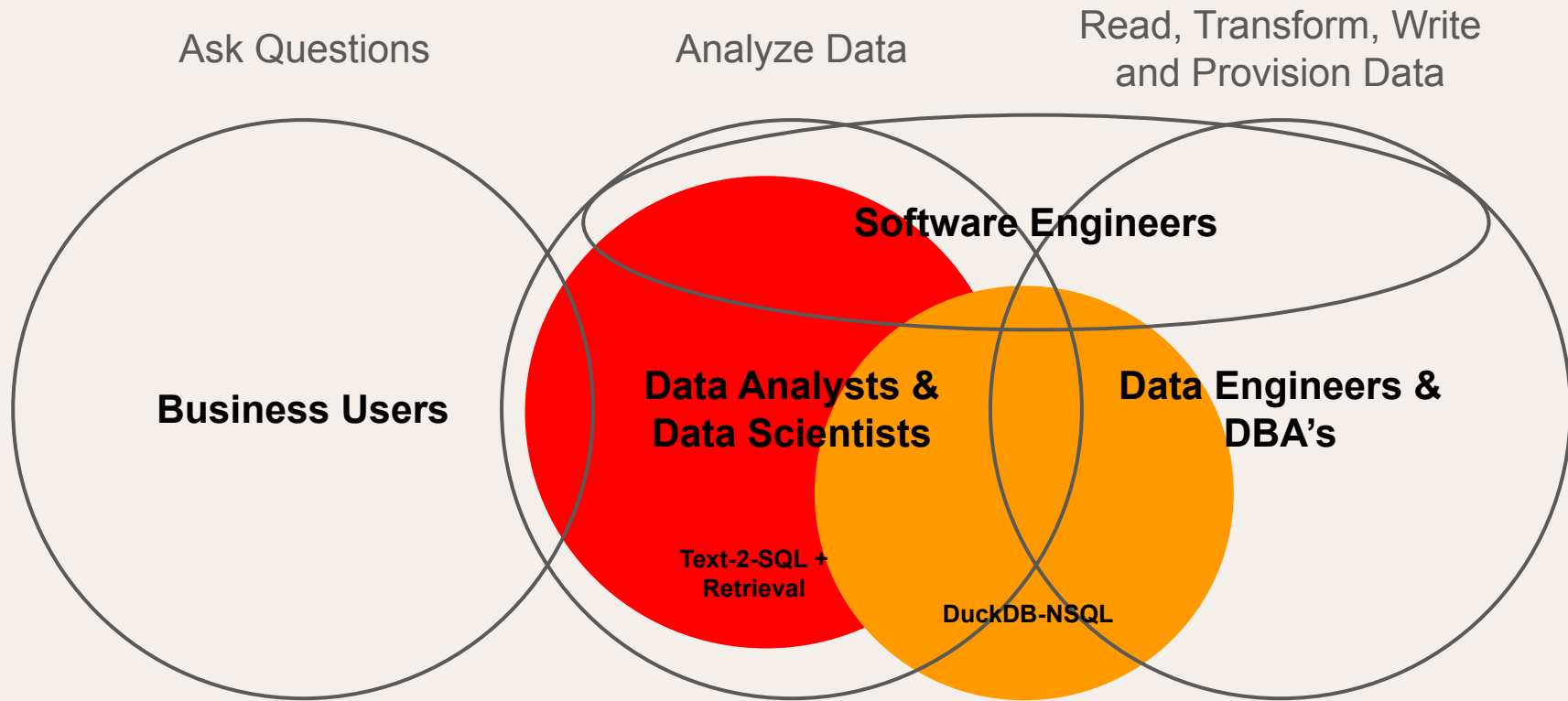
Cursor & windsurf integrate large language models to provide real-time code completion, automated refactoring, and intelligent code generation while you type.





- Drafts for Analytical Queries
- Reduce Repetitive Work
- Requires Data & SQL Knowledge for Verification





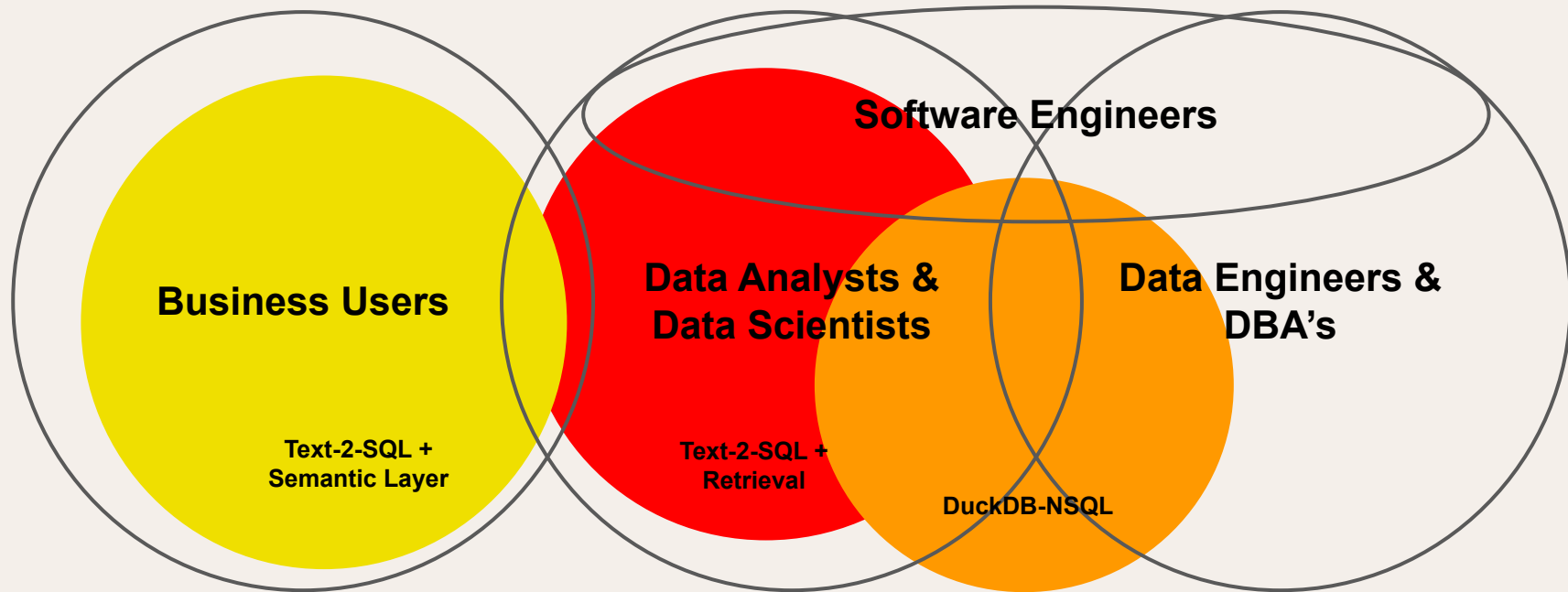
- Simple DuckDB SQL snippets for any type of statements
- Saves round trip to docs



Ask Questions

Analyze Data

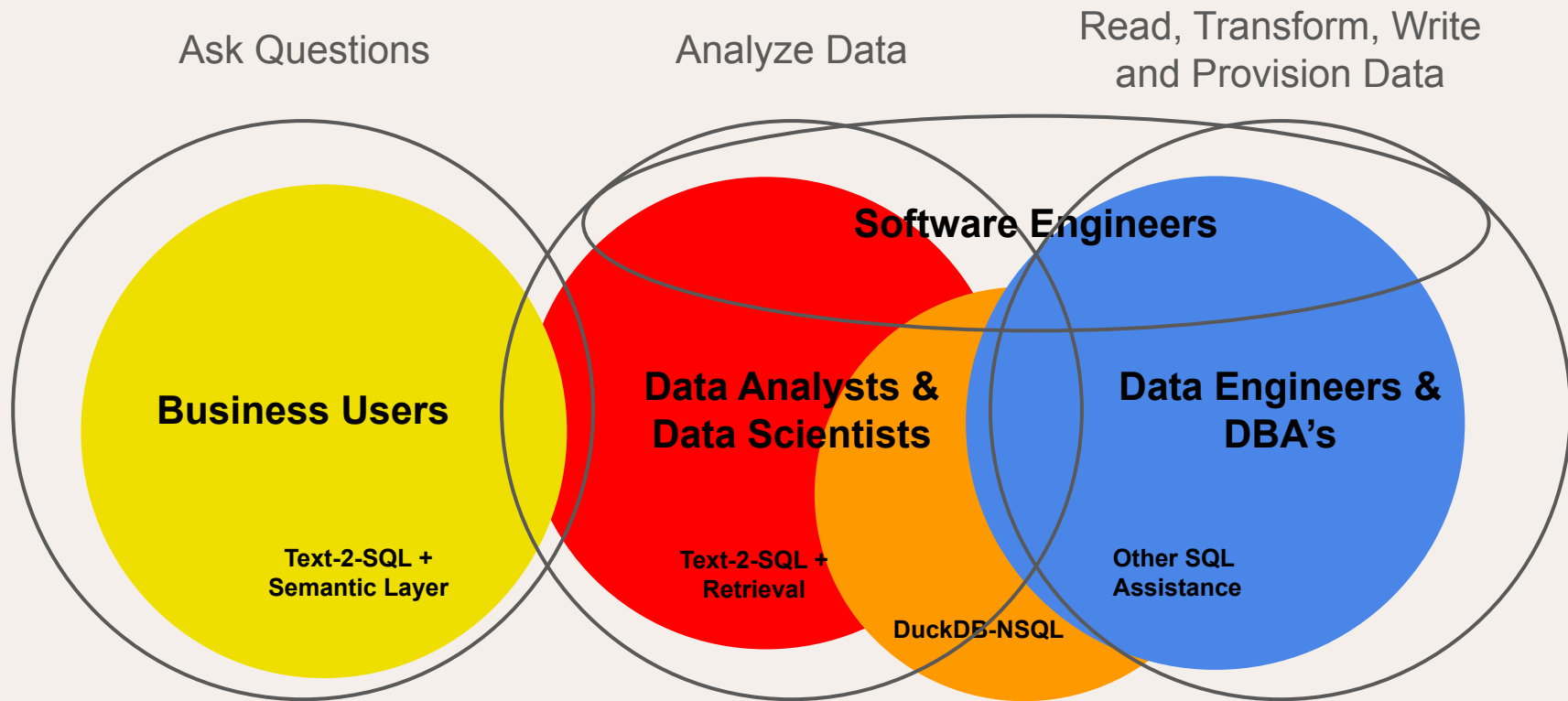
Read, Transform, Write and Provision Data



- Built-in Guardrails & Semantic Correctness
- Requires making tribal data knowledge explicit (lot of work!)

- Drafts for Analytical Queries
- Reduce Repetitive Work
- Requires Data & SQL Knowledge for Verification





- Built-in Guardrails & Semantic Correctness
- Requires making tribal data knowledge explicit (lot of work!)

- Drafts for Analytical Queries
- Reduce Repetitive Work
- Requires Data & SQL Knowledge for Verification

- IDE-Integrated
- Support for DDL / DML / ETL-Tasks
- Focus on Performance & Reliability



Upcoming webinar

Building Breakthrough AI Applications with Not Diamond and Dagster

Developer-first model routing

Improve accuracy and reduce costs with data-driven AI model recommendations.

[Book a demo](#)

[Watch a video](#)



```
from notdiamond import NotDiamond

# Define the Not Diamond routing client
client = NotDiamond()

# The best LLM is determined by Not Diamond
result, session_id, provider = client.chat.completions.create(
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Concisely explain merge sort."}
    ],
    model=['openai/gpt-4o', 'anthropic/claude-3-5-sonnet-20240620']
)

print("LLM called: ", provider.model) # The LLM routed to
print("LLM output: ", result.content) # The LLM response
```

Q&A



Thank you!