Wrangling Data with DuckDB

Will Angel

2024-10-29

Key Takeaways

1. DuckDB is a *very* fast in-process SQL database
2. Duckplyr is a package for using Dplyer on DuckDB
3. You can use Duckplyr as a *drop in replacement to get a 2-30x speedup for dplyr code on large datasets*
4. Big data is smaller than it used to be.

Agenda

- What is DuckDB
- Why should you care about DuckDB
- When should you use DuckDB
- How can you use DuckDB
- What is Duckplyr
- Data processing performance and profiling
- A brief tangent on the shrinking of big data.

DuckDB

What is DuckDB?

DuckDB is an open source fast in-process analytical database!

- Open Source: Free & Open
- Fast: Performant. **Quickly and efficiently runs analytical SQL queries**
- In-process: **Runs locally** without a server
- Analytical: DuckDB is optimized for **aggregations** and analytical queries to support online analytical processing (OLAP). DuckDB supports ACID transactions, but is not as fast for online transaction processing (OLTP) workloads
- Database: DuckDB can be used to efficiently store relational data.

Why should you care?

- DuckDB is a great tool for local SQL analysis. Import a file and you can do SQL locally!
- DuckDB is starting to power the next generation of embedded analytical tools, so expect browser based data filtering tools to get more powerful.

Why should you care (more technical)

- DuckDB is like SQLite but for data processing. You can crunch significant amounts of data locally without spinning up a full database / data warehouse server, which may create significant time/cost savings and simplify system design
- DuckDB is versatile and fast for data processing. Competitive with spark/polars in benchmarks
- DuckDB is portable with zero dependencies.

When should you use DuckDB

- You should use DuckDB for SQL data processing!
- You have medium-largish data
- You don't want the hassle of procuring larger computing resources.

How can you use DuckDB

- Directly use the DuckDB package
- use DBplyr to connect to a local DuckDB database
- use Duckplyr!

What is Duckplyr

- Duckplyr is a drop in replacement for Dplyr!
- Duckplyr uses DuckDB's "relational" API to skip the SQL and directly construct logical query plans
- This means you can speed up your Dplyr code by ~2-30x by simply changing the package!
- Unsupported operations will fall back to Dplyr, so your code will always run!
- Duckplyr *overwrites* Dplyr methods, so it only takes loading the package.

  ```
  library("duckplyr")
  ```

Duckplyr Caveats

- Unsupported operations will fall back to Dplyr, so your code will always run!
  - This may mean you don't get a performance increase
  - If you're doing out of memory processing this may also cause issues
- Duckplyr is still under active development, so there are some surprising unsupported operations:
  - "message":"No relational implementation for group_by()."

Benchmarking & Speed

Dplyr versus Duckplyr

To compare Dplyr and Duckplyr performance, we'll look at the Global Lake area, Climate, and Population Dataset (GLCP). This includes almost 80 million records of temperature data for lakes, for a 3.9GB CSV file.

Dplyr:

```
data |>
  distinct(HYBAS_ID) |>
  nrow()
```

Duckplyr:

```
data |>
  duckplyr::as_duckplyr_tibble() |>
  distinct(HYBAS_ID) |>
  nrow()
```

Dplyr Versus Duckplyr

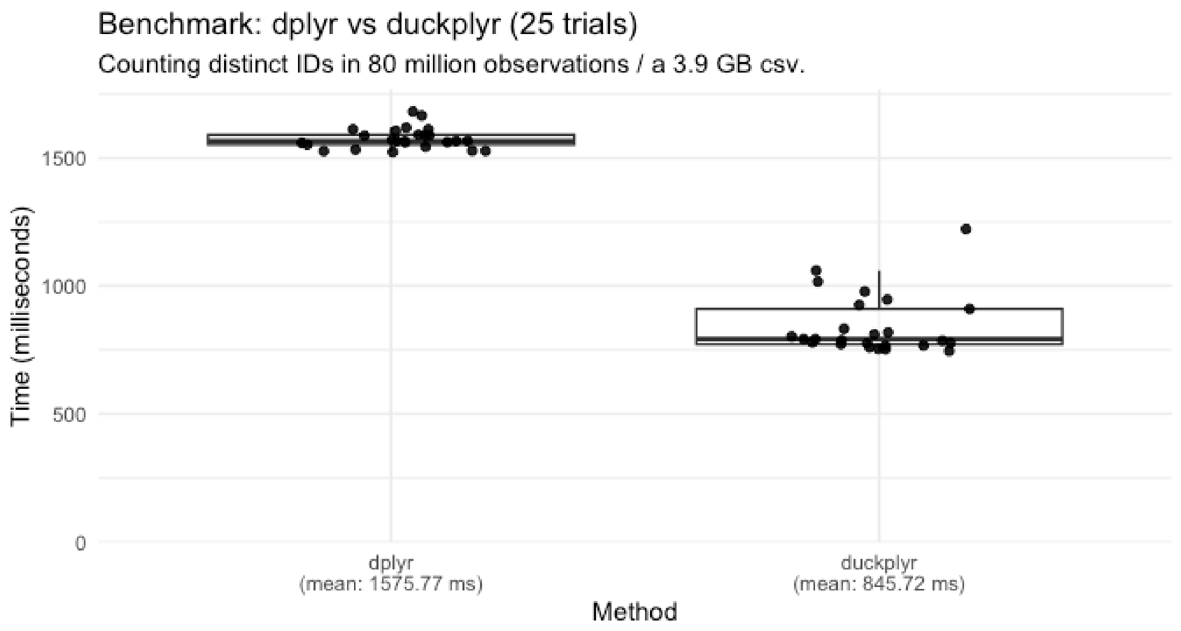


Figure 1: 86% speed increase counting distinct IDs in 80 million rows

Dplyr Versus Duckplyr

Now we'll take a look at the overall GLCP dataset, 30 million observations in a 5.5GB CSV file.

```
data |>
  summarise(
    avg_size = mean(total_km2), n = n(), .by = country)
```

```
library(duckplyr)

data |>
  summarise(
    avg_size = mean(total_km2), n = n(), .by = country)
```

Dplyr Versus Duckplyr

Dplyr Versus ~~Duckplyr~~ DuckDB

Let's switch to DuckDB and write some SQL:

```
library(duckdb)

con <- dbConnect(duckdb())
duckdb_register(con, "data", data)
```

```
dbGetQuery(con, "
      SELECT
        country,
        AVG(total_km2) as avg_size,
        COUNT(*) as n
      FROM data
      GROUP BY country
    ")
```

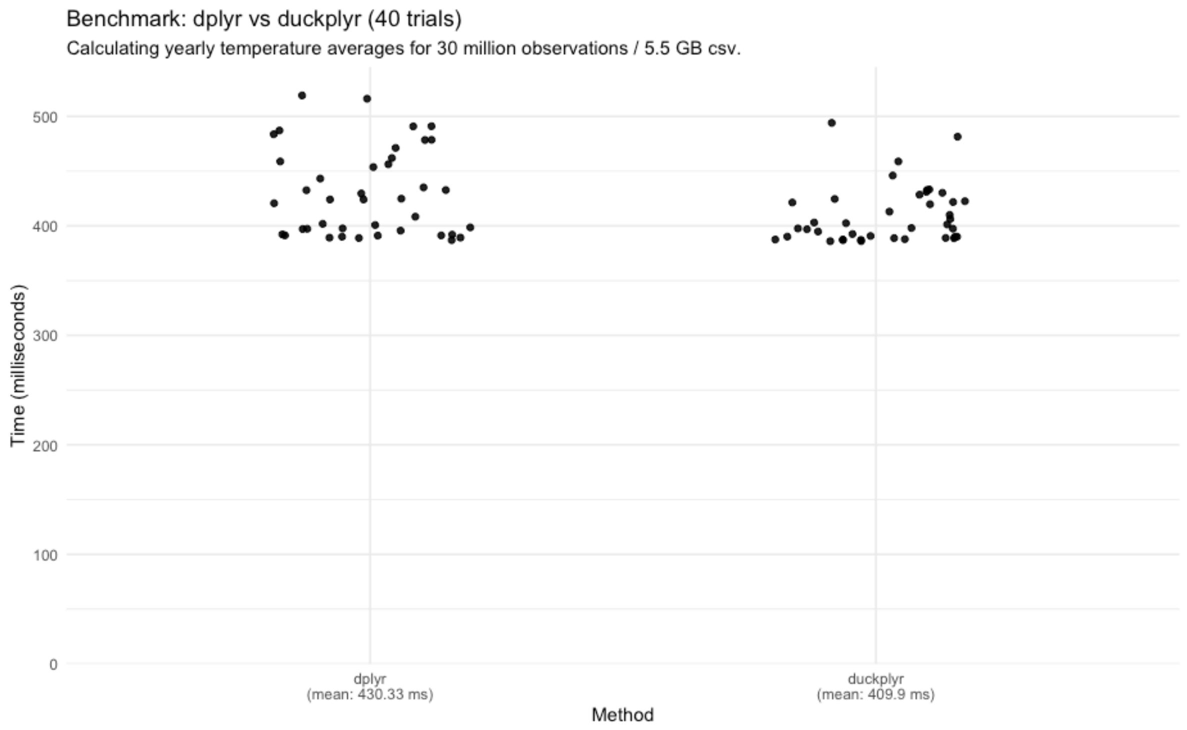


Figure 2: We ran into a limitation of Duckplyr and fell back to dplyr. “{”version”:“0.4.1”,“message”:“No relational implementation for group_by()”}”

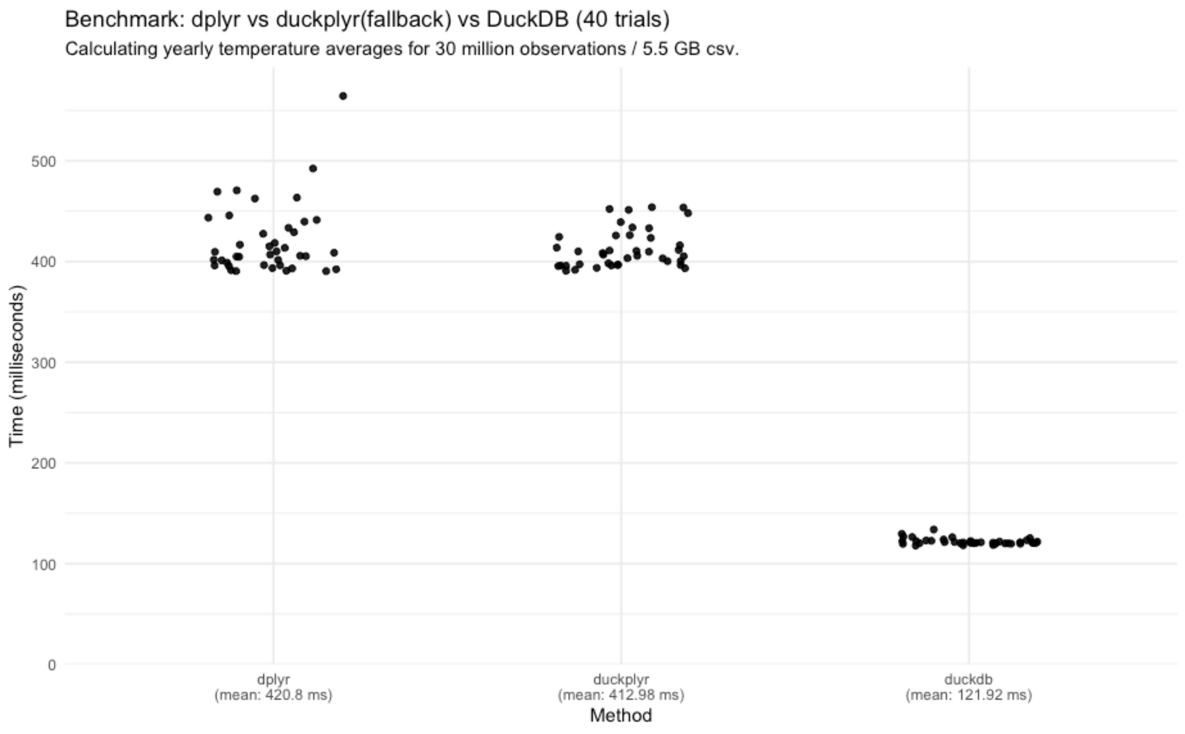


Figure 3: And now we can get results much faster

Dplyr Versus ~~Duckplyr~~ DuckDB

Big Data is smaller than it used to be

Computers are getting bigger

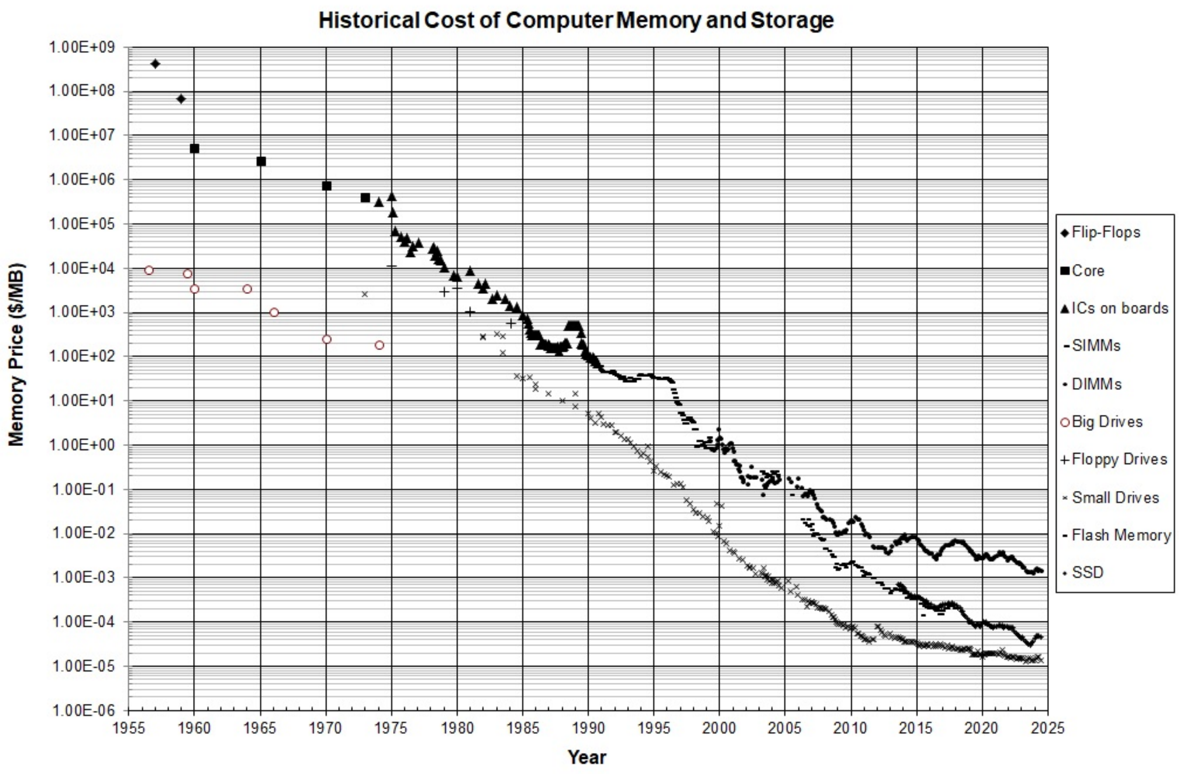


Figure 4: https://jcmit.net/memoryprice.htm

Computers are getting bigger

Memory has gotten significantly cheaper

Memory has gotten significantly cheaper

In Conclusion:

Big data is smaller than it used to be

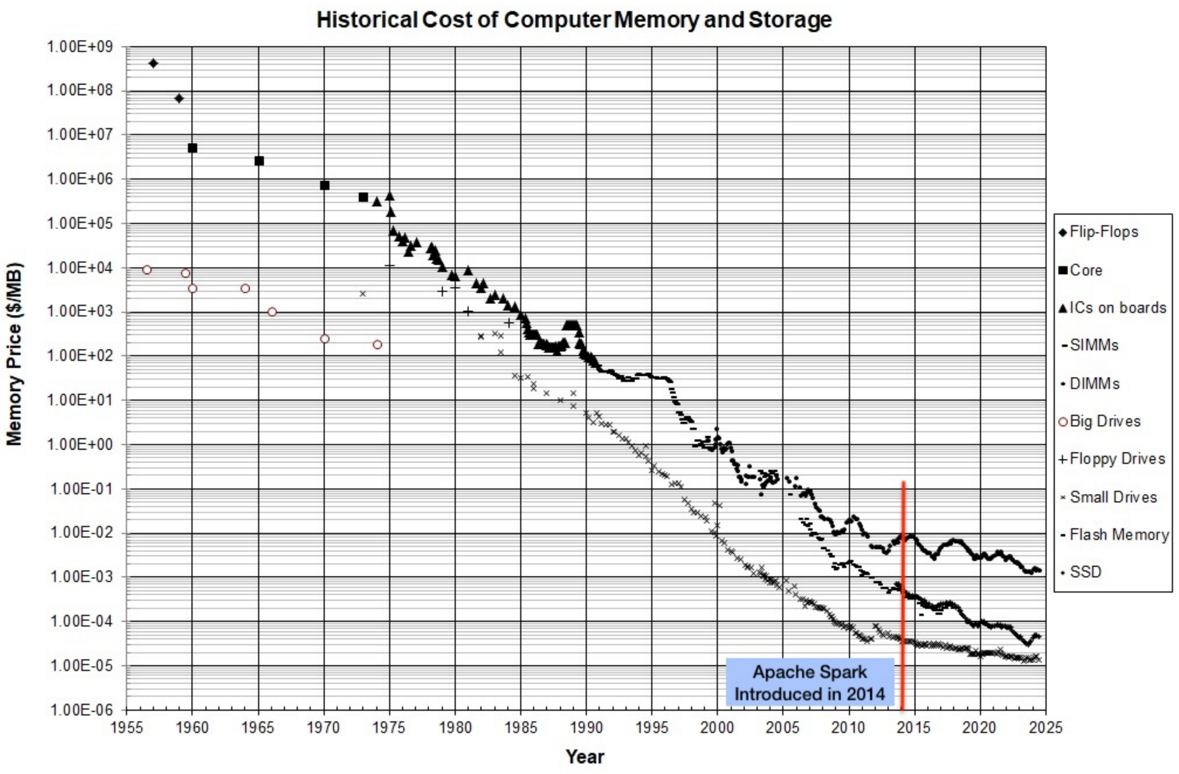


Figure 5: https://jcmit.net/memoryprice.htm

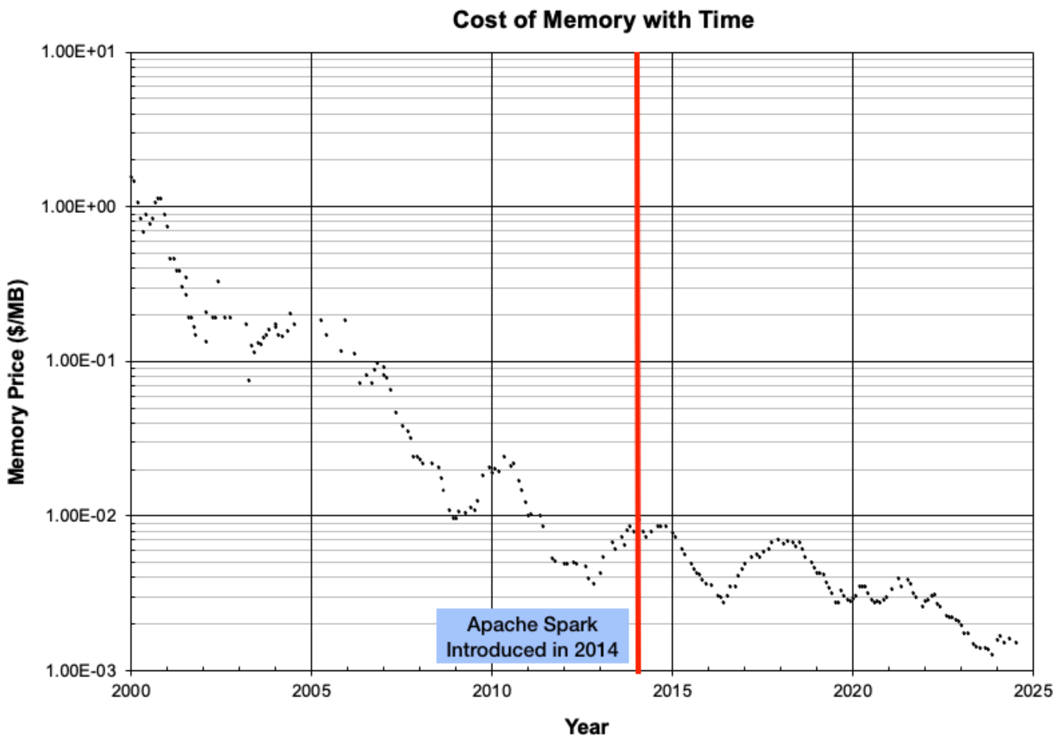


Figure 6: https://jcmit.net/memoryprice.htm

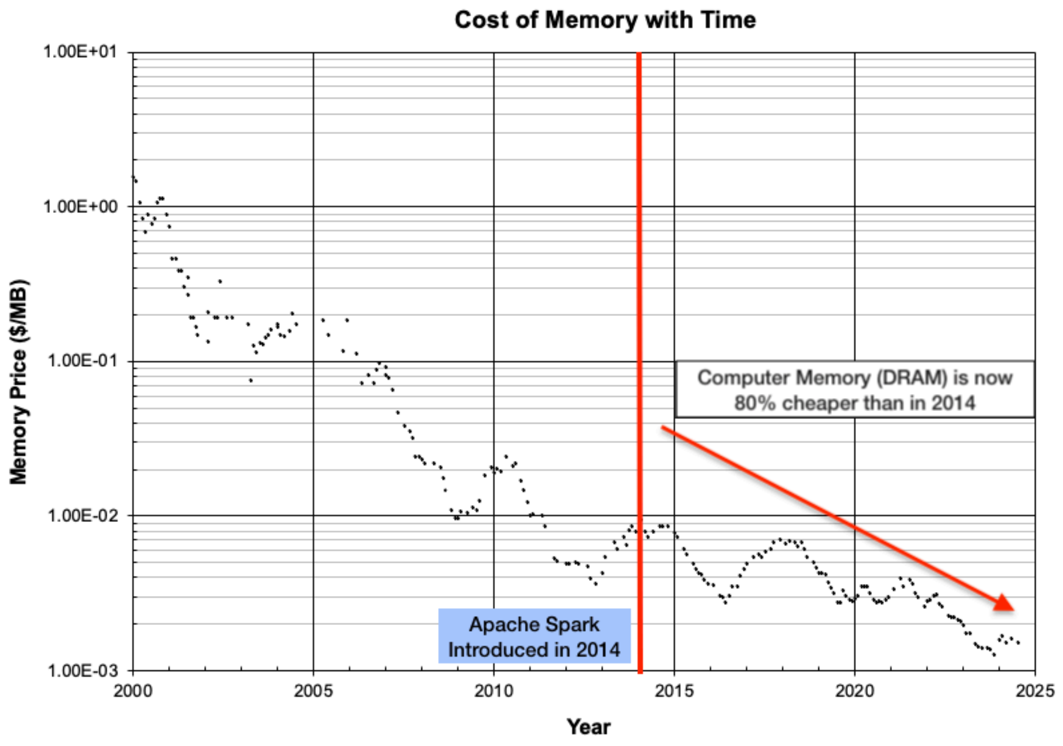
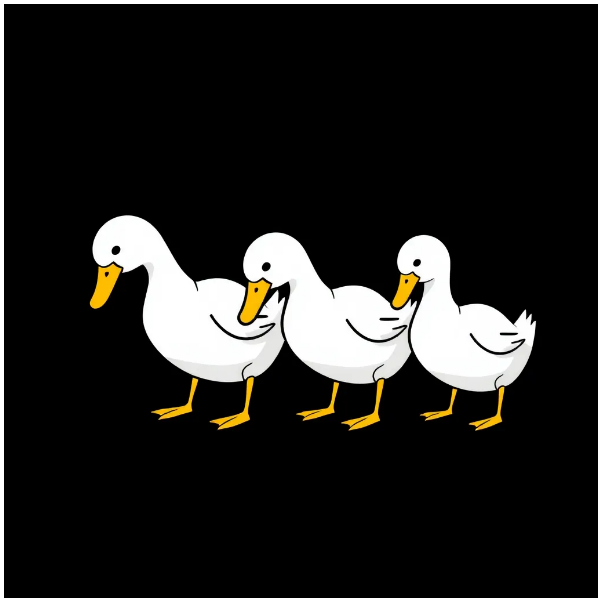


Figure 7: https://jcmit.net/memoryprice.htm

With more access to cheaper computing power and better tools (DuckDB/Polars(TidyPolars)/Data.Table) that allow us to process data more efficiently:

1. We'll end up with more "Annoyingly Medium Data" - Too big to open in Excel, fits in RAM on a decent laptop
2. We can defer investing in large scale distributed systems like Spark more often, and wait longer to ditch our dplyr code.

Thank you!


AI ducks taking a bow: Flux-schnell Image Generation model.

- /in/william-angel/
- @datadrivenangel
- www.williamangel.net
- yt@williamangel.net