Contributions:

- Trayna:
  - Built the formatting of our directory/repository
  - Implemented MySQL schema and indexes
  - Implemented MySQL API layer (twitter_mysql.py)
  - Implemented DB connection utilities (dbutils.py)
  - Debugged SQL queries using MySQL Workbench
  - Helped with documentation/analysis
- Emily:
  - Implemented CSV loaders (load_follows.py, load_tweets.py)
  - Implemented benchmarking driver (timeline_test.py)
    - random user selection
  - Ran experiments and collected performance numbers
  - Helped with documentation/analysis

Analysis & Reporting

Hardware Configuration:

- Machine: MacBook Pro
- CPU: Apple M4 (10 cores)
- RAM: 16 GB
- Storage: 512 GB SSD
- OS: macOS Sequoia (15.5)

Software Stack:

- Programming Language: Python
- Relational DB: MySQL (9.6.0)
- Libraries: mysql-connector-python(==9.5.0), python-dotenv, pandas
- Tools: Visual Studio Code, MySQL WorkBench, Git/GitHub

Result Table:

API Function	**API Function Calls/Sec**
postTweet	4559.17 postTweet calls/sec (Inserted 1,000,000 tweets in 219.34s)
getHomeTimeline	573.90 calls/sec (50,000 getHomeTimeline calls in 87.12s)

Otimizations/Observations:

- Index choices: Adding indexes on *user_id, tweet_ts*, and *follower_id* had a big impact on performance, since they reduced the need for full table scans and made lookups and joins much faster.
- Hardware limitations: Performance was also influenced by the hardware on my laptop, specifically the available RAM (16 GB) and SSD storage. These affect how much data can be kept in memory versus having to be read from disk.

- Background activity: Running other applications at the same time may have affected CPU and memory availability, which could slightly impact the timing results.

Note:
ChatGPT was used as support to help with interpreting error messages, exploring alternative approaches, as well as general guidance and checking my understanding of certain steps.