MLtraq: Track your AI experiments at HYPERSPEED

Michele Dallachiesa
Data Products & AI Consulting
michele.dallachiesa@sigforge.com

Scope of this talk

You will learn:

- What is experiment tracking
- What makes different frameworks fast and slow
- How to select an experiment tracker for your projects

https://mltraq.com/benchmarks/speed/

Experimentation

- Definition: “The process of systematically changing and testing different input values in an algorithm to observe their impact on performance, behavior, or outcomes.”

Experiment tracking

- Definition: “The process of recording the inputs, outputs, and performance metrics of an experiment.”
- Examples: Code, notebooks, scripts, environment setup, parameters, configurations, evaluation metrics, model weights, system stats, inputs, outputs, accuracy, prompts, cost metadata, ...

Applications of experiment tracking

- Explore and understand the impact on performance of different algorithms, parameters, and datasets
- Automation and observability: live monitoring of long-term experiments, reproducibility, documentation, collaboration, ...

Modelling experiments

- An experiment is a collection of runs
- A run is an instantiation of the experiment with a fixed set of inputs

Why tracking speed matters: Initialization (1/3)

- Slow imports negatively impact development, CI/CD tests, and debugging speed
- High run initialization times impact on our ability to experiment with hundreds of thousands of runs

Wouldn't it be nice to start tracking almost instantly?

Why tracking speed matters: High frequency (2/3)

- At times, it's necessary to record metrics that occur frequently (loss, reward, state, ...)
- Workarounds to handle too much information come at a complexity/completeness/accuracy cost: threading, downsampling, summarization, and histograms

What if we could avoid these limitations altogether?

Why tracking speed matters: Large, complex objects (3/3)

- Python data structures (dictionaries, lists, tuples), NumPy arrays, data frames, datasets, model weights, timeseries, forecasts, media files such as images, audio recordings, and videos, ...
- Existing solutions are primitive and slow, using tech (JSON, uuencoding) from 25-40 years ago

What if we could track more with less constraints?

- A new open-source experiment tracker designed to work with any SQL database, fast and interoperable
- Serialization powered by native SQL database types, Numpy, PyArrow, and safe Python pickles
- Funding: You can star the project on GitHub and/or hire me to make your experiments run faster

Tracking an experiment

```python
from mltraq import create_session

session = create_session("sqlite:///local.db")
experiment = session.create_experiment("test")

with experiment.run() as run:
    run.fields.accuracy = .8

with experiment.run() as run:
    run.fields.accuracy = .9

experiment.persist(if_exists="replace")

session.load_experiment("test").runs.df()
```

	id_run	accuracy
0	8c7a7584-6037-4187-9656-c45229db874d	0.8
1	1304b76a-4cd0-4990-a366-13081e128f3b	0.9

Benchmarking experiment tracking frameworks

Frameworks

- Weights & Biases (0.16.3)
- MLflow (2.11.0)
- FastTrackML (0.5.0b2)
- Neptune (1.9.1)
- Aim (3.18.1)
- Comet (3.38.1)
- MLtraq (0.0.125)

Latest update: 2024.03.06

Varying

- Value type: float, ndarray
- Count of values
- Count of runs
- Array length

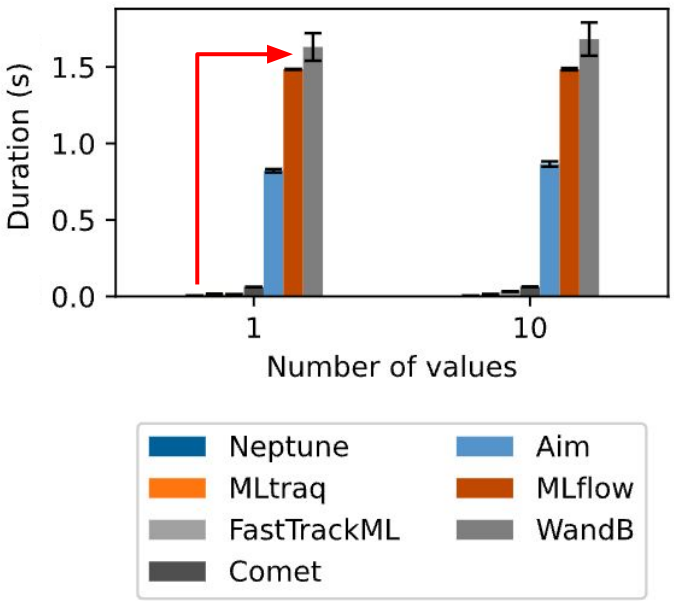
How

- As MLtraq experiments!
- 10 independent runs
- Local storage

How much time to track 1 run and 1 value?

Start up time

Neptune vs W&B: 400x



What takes most of the time?

- W&B: threading, IPC
- MLflow: Alembic migration
- Aim: threading, RocksDB
- Comet: threading
- FastTrackML: fast but requires running server
- MLtraq: SQLite operations
- Neptune: direct writes to FS

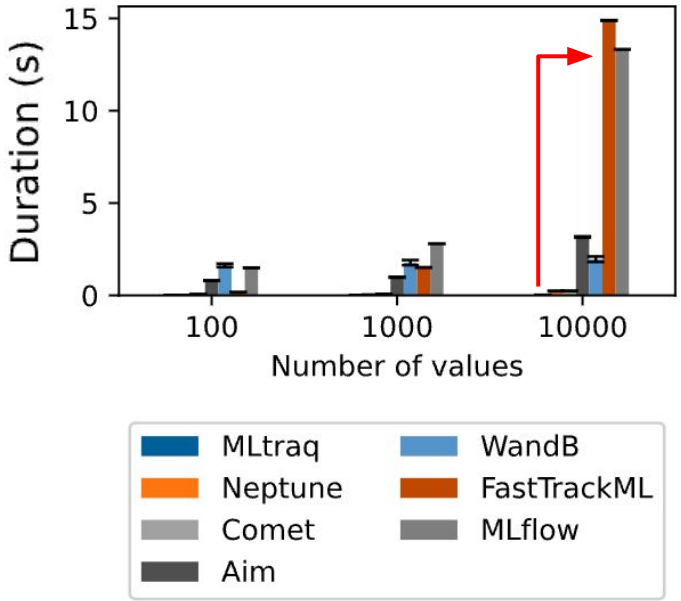
MLflow: Alembic migration

```
2024/04/09 12:08:59 INFO mlflow.store.db.utils: Updating database tables
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
INFO  [alembic.runtime.migration] Running upgrade  -> 451aebb31d03, add metric step
INFO  [alembic.runtime.migration] Running upgrade 451aebb31d03 -> 90e64c465722, migrate user column to tags
INFO  [alembic.runtime.migration] Running upgrade 90e64c465722 -> 181f10493468, allow nulls for metric values
INFO  [alembic.runtime.migration] Running upgrade 181f10493468 -> df50e92ffc5e, Add Experiment Tags Table
INFO  [alembic.runtime.migration] Running upgrade df50e92ffc5e -> 7ac759974ad8, Update run tags with larger limit
INFO  [alembic.runtime.migration] Running upgrade 7ac759974ad8 -> 89d4b8295536, create latest metrics table
INFO  [89d4b8295536_create_latest_metrics_table_py] Migration complete!
INFO  [alembic.runtime.migration] Running upgrade 89d4b8295536 -> 2b4d017a5e9b, add model registry tables to db
INFO  [2b4d017a5e9b_add_model_registry_tables_to_db_py] Adding registered_models and model_versions tables to database.
INFO  [2b4d017a5e9b_add_model_registry_tables_to_db_py] Migration complete!
INFO  [alembic.runtime.migration] Running upgrade 2b4d017a5e9b -> cfd24bdc0731, Update run status constraint with killed
INFO  [alembic.runtime.migration] Running upgrade cfd24bdc0731 -> 0a8213491aaa, drop_duplicate_killed_constraint
INFO  [alembic.runtime.migration] Running upgrade 0a8213491aaa -> 728d730b5ebd, add registered model tags table
INFO  [alembic.runtime.migration] Running upgrade 728d730b5ebd -> 27a6a02d2cf1, add model version tags table
INFO  [alembic.runtime.migration] Running upgrade 27a6a02d2cf1 -> 84291f40a231, add run_link to model_version
INFO  [alembic.runtime.migration] Running upgrade 84291f40a231 -> a8c4a736bde6, allow nulls for run_id
INFO  [alembic.runtime.migration] Running upgrade a8c4a736bde6 -> 39d1c3be5f05, add_is_nan_constraint_for_metrics_tables_if_necessary
INFO  [alembic.runtime.migration] Running upgrade 39d1c3be5f05 -> c48cb773bb87, reset_default_value_for_is_nan_in_metrics_table_for_mysq
INFO  [alembic.runtime.migration] Running upgrade c48cb773bb87 -> bd07f7e963c5, create index on run_uuid
INFO  [alembic.runtime.migration] Running upgrade bd07f7e963c5 -> 0c779009ac13, add deleted_time field to runs table
INFO  [alembic.runtime.migration] Running upgrade 0c779009ac13 -> cc1f77228345, change param value length to 500
INFO  [alembic.runtime.migration] Running upgrade cc1f77228345 -> 97727af70f4d, Add creation_time and last_update_time to experiments tab
e
INFO  [alembic.runtime.migration] Running upgrade 97727af70f4d -> 3500859a5d39, Add Model Aliases table
INFO  [alembic.runtime.migration] Running upgrade 3500859a5d39 -> 7f2a7d5fae7d, add datasets inputs input_tags tables
INFO  [alembic.runtime.migration] Running upgrade 7f2a7d5fae7d -> 2d6e25af4d3e, increase max param val length from 500 to 8000
INFO  [alembic.runtime.migration] Running upgrade 2d6e25af4d3e -> acf3f17fdcc7, add storage location field to model versions
INFO  [alembic.runtime.migration] Context impl SQLiteImpl.
INFO  [alembic.runtime.migration] Will assume non-transactional DDL.
```

How much time to track 1 run and 100-10K values?

High frequency tracking

MLtraq vs MLflow: 355x



- Entity-attribute-value database model with no batching kills MLflow/FastTrackML performance

Attribute			
Id	Entity	Name	isNum
1	1	Capacity, GB	1
2	1	Endurance, TWB	1
3	3	“accuracy”	0

Entity	
Id	Name
1	SSD
2	HDD

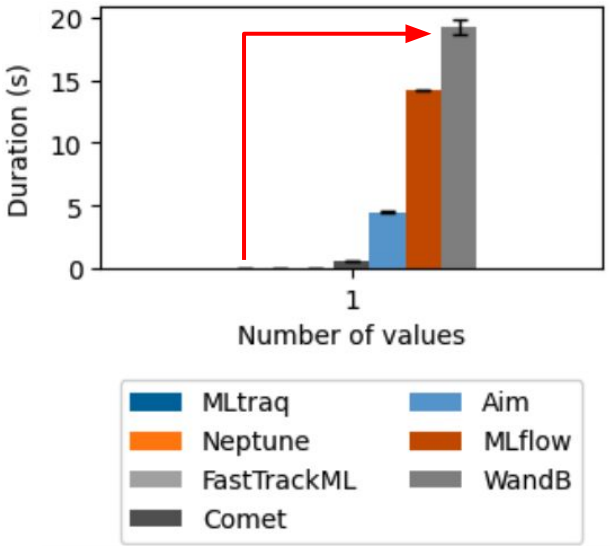
Experiment ID/name

Value				
Instance_id	Entity_id	Attribute_id	Text_value	Num_value
1	1	1		480
2	2	1	0.85	2000
1	1	2		1800
3	3	1		800
3	3	3	silver	

Source: https://community.intersystems.com/post/entity-attribute-value-model-relational-databases-should-globals-be-emulated-tables-part-1

How much time to track 10 runs and 1 value?

MLtraq vs W&B: 1563x

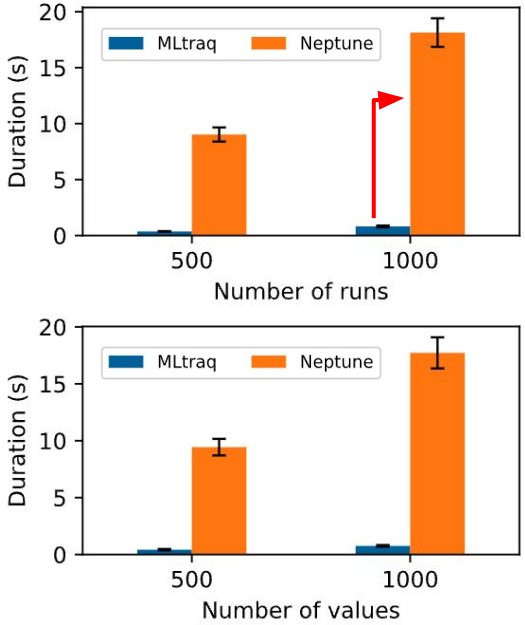


- Threading/IPC expensive for W&B

```
24.740 test_wandb                                   ../../../../../var/folders/2c/7zw
    23.220 init
      ▾ 47 frames hidden (wandb, threading, <built-in>...)
        22.396 _WandbInit.init
            13.383 Run.wrapper
                13.383 Run.wrapper
                    13.383 Run.finish
                        13.380 Run._finish
                            12.947 Run._atexit_cleanup
                                12.941 Run._on_finish
                                    9.018 Backend.cleanup
                                        9.018 InterfaceSock.join
                                            9.015 MessageSockRouter.join
                                                9.015 Thread.join
                                                    9.015 Thread._wait_for_tstate_lock
                                                        9.015 lock.acquire
```

How much time to track 1K runs and 1K values?

MLtraq vs Neptune: 23x



What makes MLtraq faster

- SQLite vs filesystem
- Safe pickling vs JSON

SQLite

Small. Fast. Reliable.
Choose any three.

Home About Documentation Download License Support Purchase Search

35% Faster Than The Filesystem

▶ Table Of Contents

1. Summary

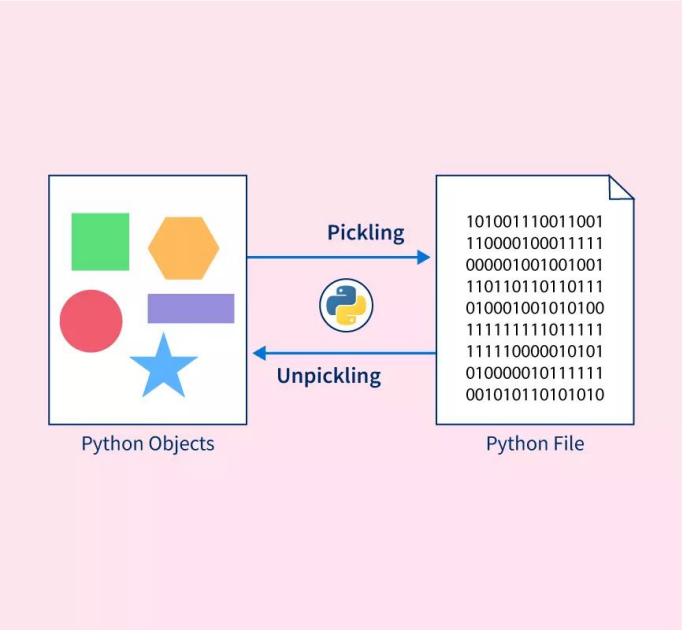
SQLite reads and writes small blobs (for example, thumbnail images) 35% faster[1] than the same blobs can be read from or written to individual files on disk using fread() or fwrite().

Furthermore, a single SQLite database holding 10-kilobyte blobs uses about 20% less disk space than storing the blobs in individual files.

The performance difference arises (we believe) because when working from an SQLite database, the open() and close() system calls are invoked only once, whereas open() and close() are invoked once for each blob when using blobs stored in individual files. It appears that the overhead of calling open() and close() is greater than the overhead of using the database. The size reduction arises from the fact that individual files are padded out to the next multiple of the filesystem block size, whereas the blobs are packed more tightly into an SQLite database.

Source: https://www.sqlite.org/fasterthanfs.html

Python pickles harmless if limited to safe opcodes



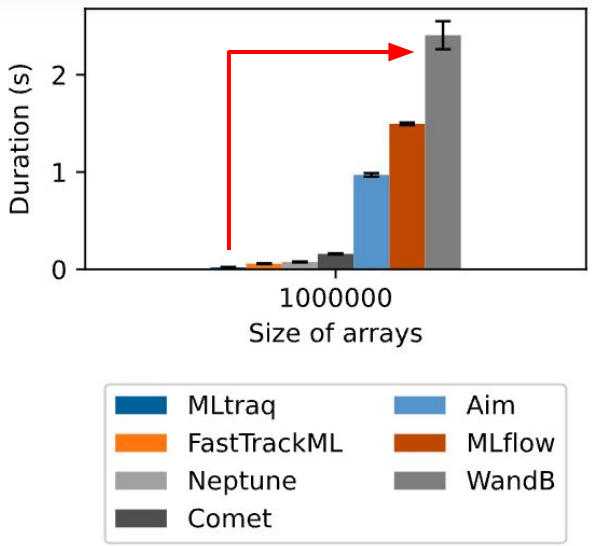

```
┌──(kali㉿kali)-[~/ctf/pickle]
└─$ python3 -m pickletools backup.data
    0: \x80 PROTO      4
    2: \x95 FRAME      42
   11: \x8c SHORT_BINUNICODE 'posix'
   18: \x94 MEMOIZE    (as 0)
   19: \x8c SHORT_BINUNICODE 'system'
   27: \x94 MEMOIZE    (as 1)
   28: \x93 STACK_GLOBAL
   29: \x94 MEMOIZE    (as 2)
   30: \x8c SHORT_BINUNICODE 'cat /etc/passwd'
   47: \x94 MEMOIZE    (as 3)
   48: \x85 TUPLE1
   49: \x94 MEMOIZE    (as 4)
   50: R    REDUCE
   51: \x94 MEMOIZE    (as 5)
   52: .    STOP
highest protocol among opcodes = 4

┌──(kali㉿kali)-[~/ctf/pickle]
└─$
```

Source: https://infosecwriteups.com/vulnerabilities-in-python-serialization-pickle-d2385de642f6 and https://www.scaler.com/topics/pickle-python/

How much time to track 10^6 float64 values (8MB)?

Tracking large objects

MLtraq vs W&B: 113x

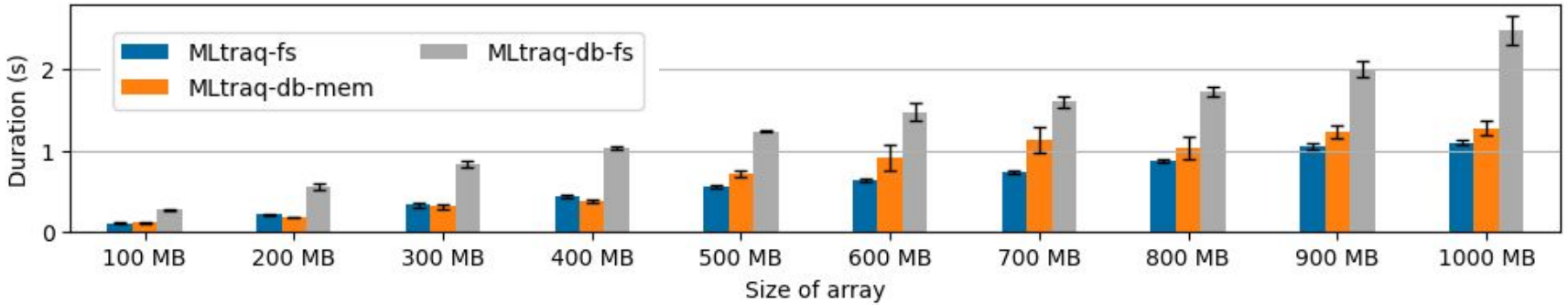


- MLtraq: Pickle, numpy.lib.format
- W&B: wandb.Table, JSON format
- Neptune: JSON, uuencoded binary blob
- MLflow: mlflow.log_text, binary blob
- FastTrackML: c.log_text, binary blob
- Aim: run.track, binary blob
- Comet: run.log_text, binary blob

binary blob = weak semantics!

How much time to track up to 10^9 int8 values (1GB)?

- Write speed of `np.zeros(size, dtype=np.int8)`
- Variants: MLtraq-fs vs MLtraq-db-mem vs MLtraq-db-fs



Conclusion

- Trade-offs: threading/IPC, data storage design, batching vs streaming
- Uuencoding and JSON-like formats are slow with poor semantics, the future is native types with PyArrow
- Beyond “tracking speed”: backward compatibility, cloud, backend, third-party integrations, reporting, complete model lifecycle management, ...
- Disclaimer: lots of simplifications in these slides, check out full article and notebooks for details!

Thank You!

Michele Dallachiesa
Data Products & AI Consulting
michele.dallachiesa@sigforge.com