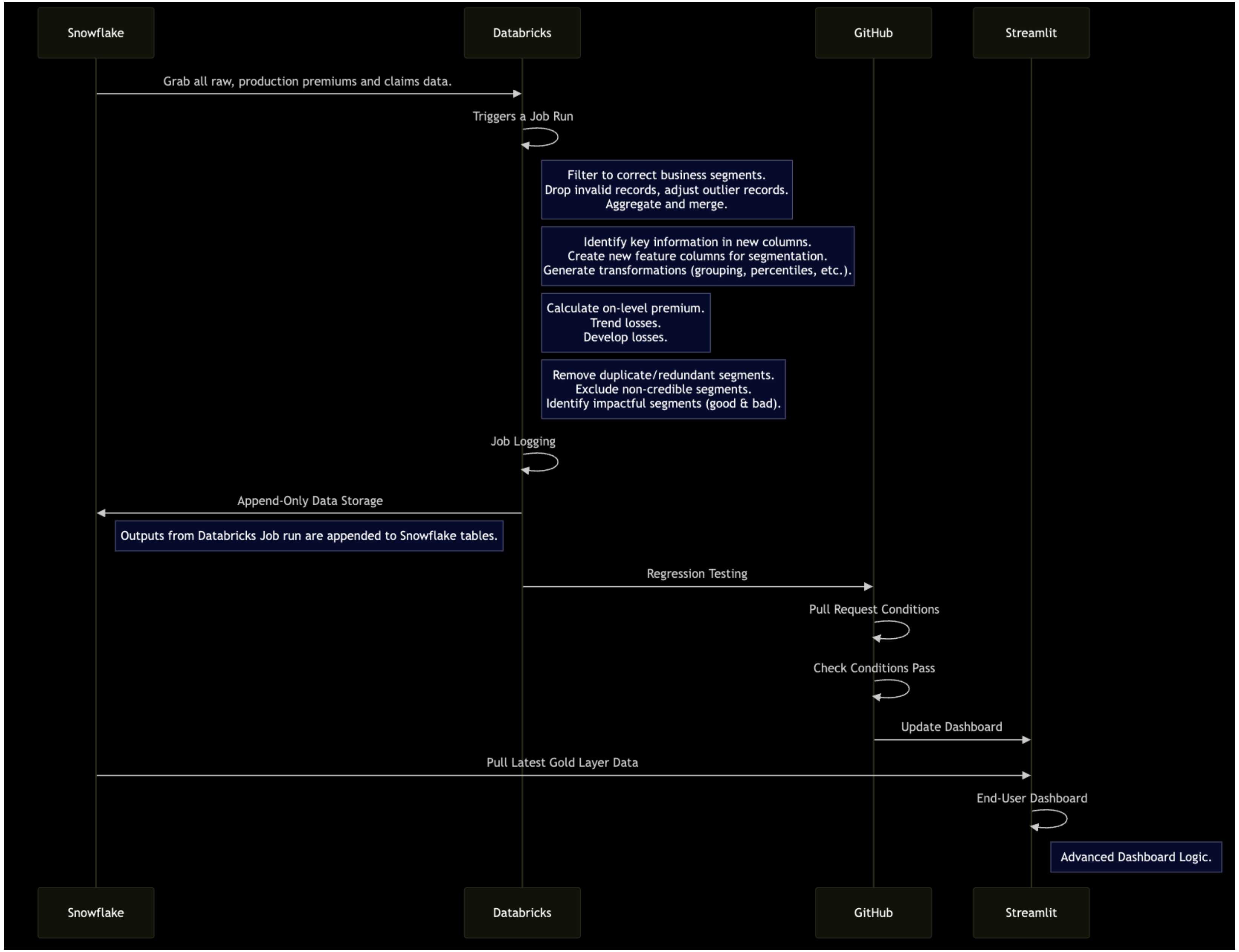
Youth VILLAGES
The force for families

Second
PRESBYTERIAN CHURCH

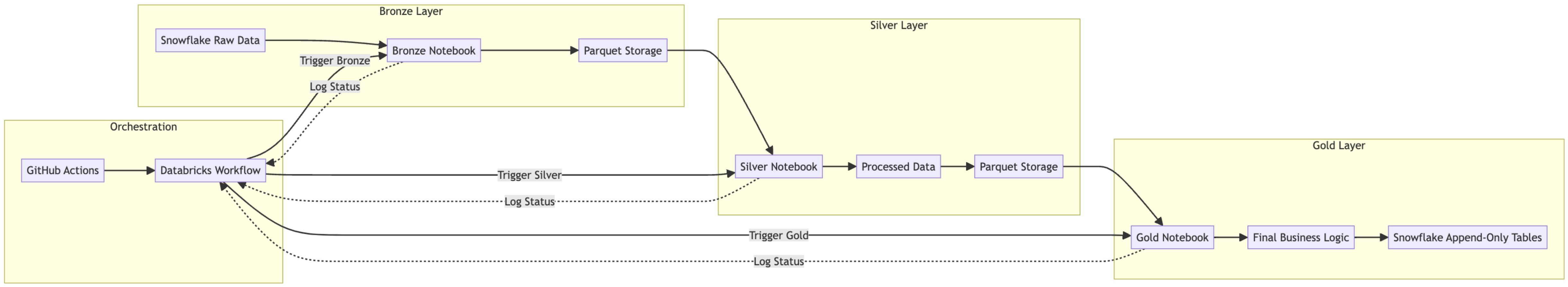
CBU

About Me

Sample Sequence Diagram of E2E Architecture

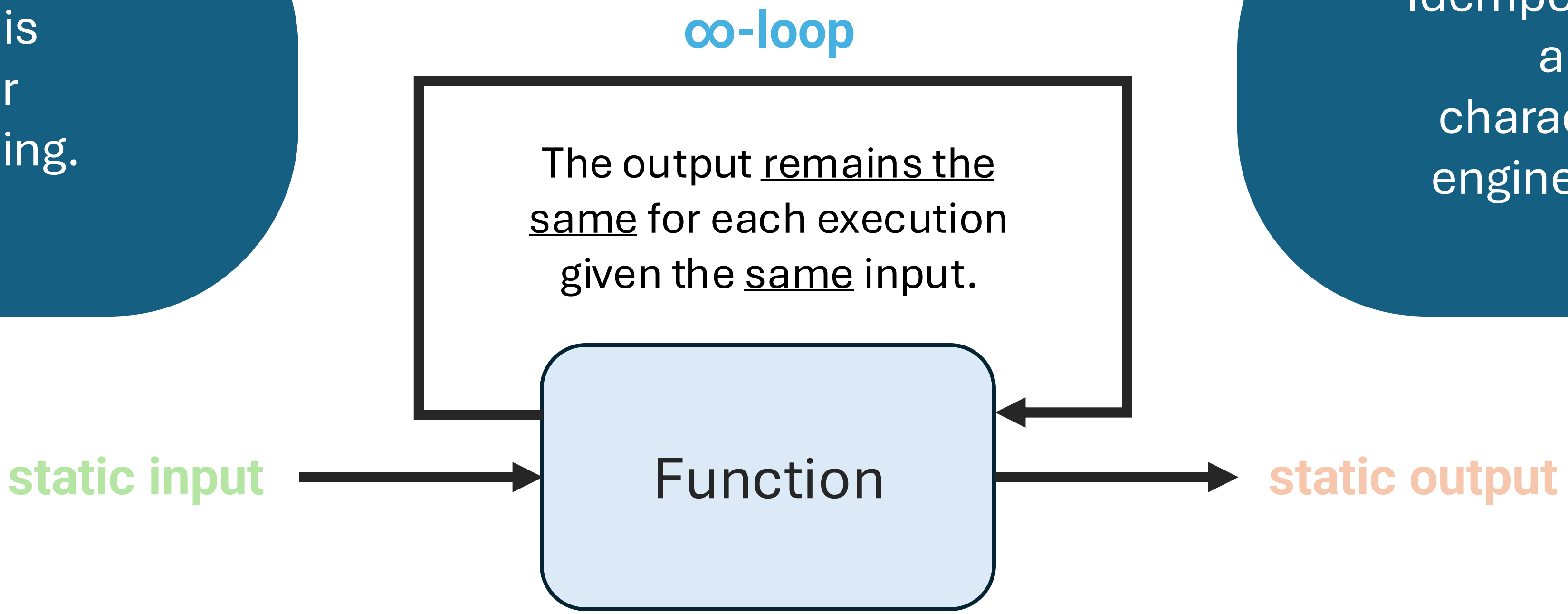


Sample Flow Chart of Medallion Architecture



Idempotency is necessary for application testing.

Idempotency



Idempotency is typically an essential characteristic of data engineering pipelines.

Non-Idempotent

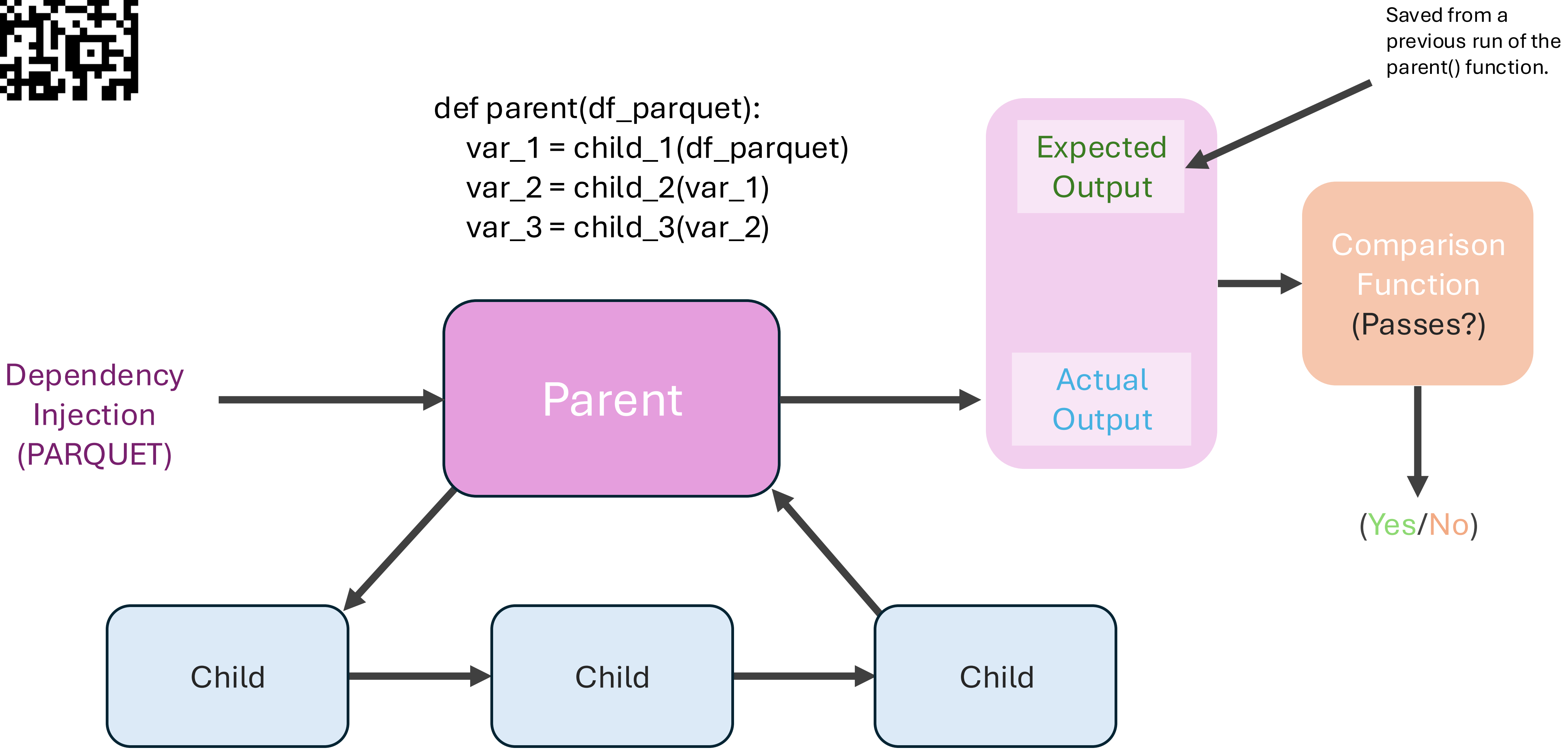
```
def func(float_val):
    float_val += random_int()
    return float_val
```

Idempotent

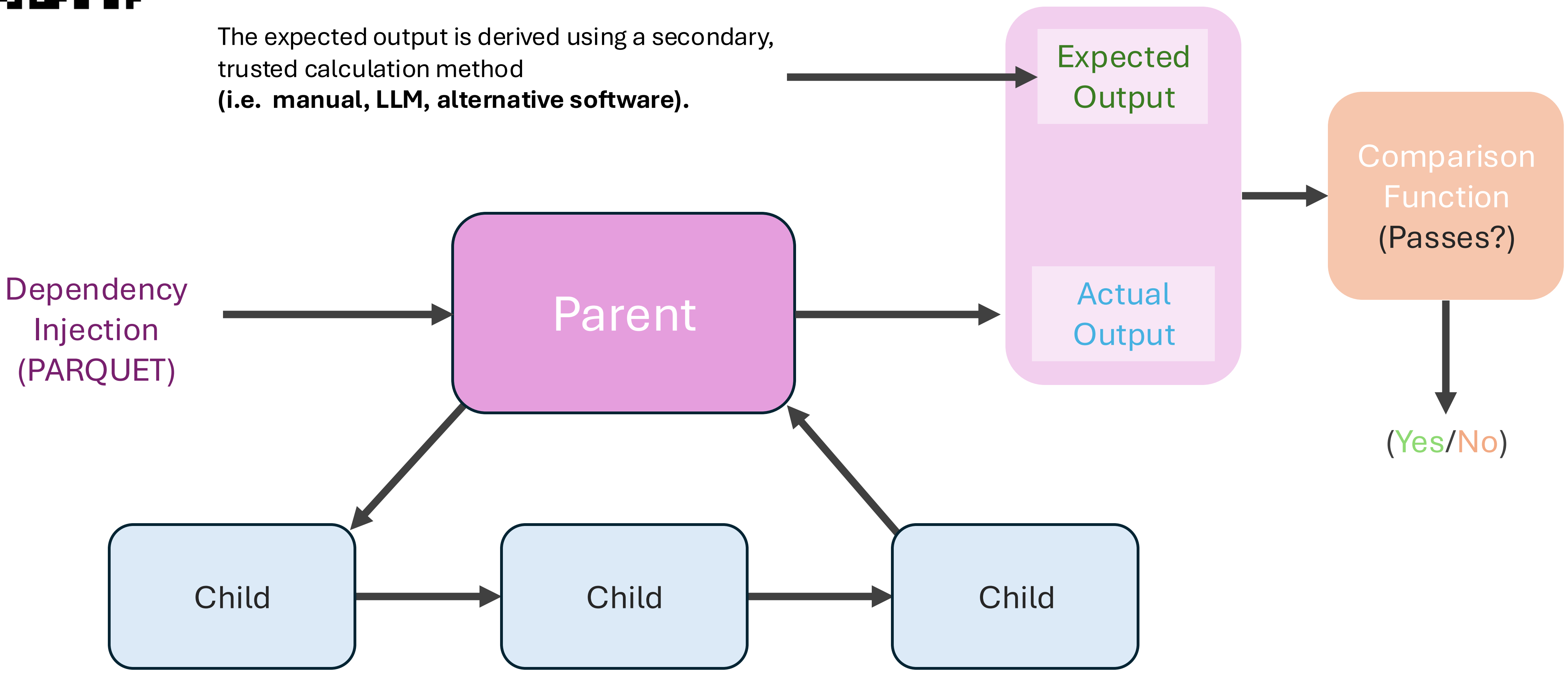
```
def func(float_val):
    return float_val
```

Integration Snapshot Testing (Analytics Engineering)

```
def parent(df_parquet):
    var_1 = child_1(df_parquet)
    var_2 = child_2(var_1)
    var_3 = child_3(var_2)
```



Integration Unit Testing



Alternative Testing

"Good software is well-tested software."

1. SMOKE TESTING

Does it run?

2. PERFORMANCE TESTING

Is it fast?

3. UI TESTING

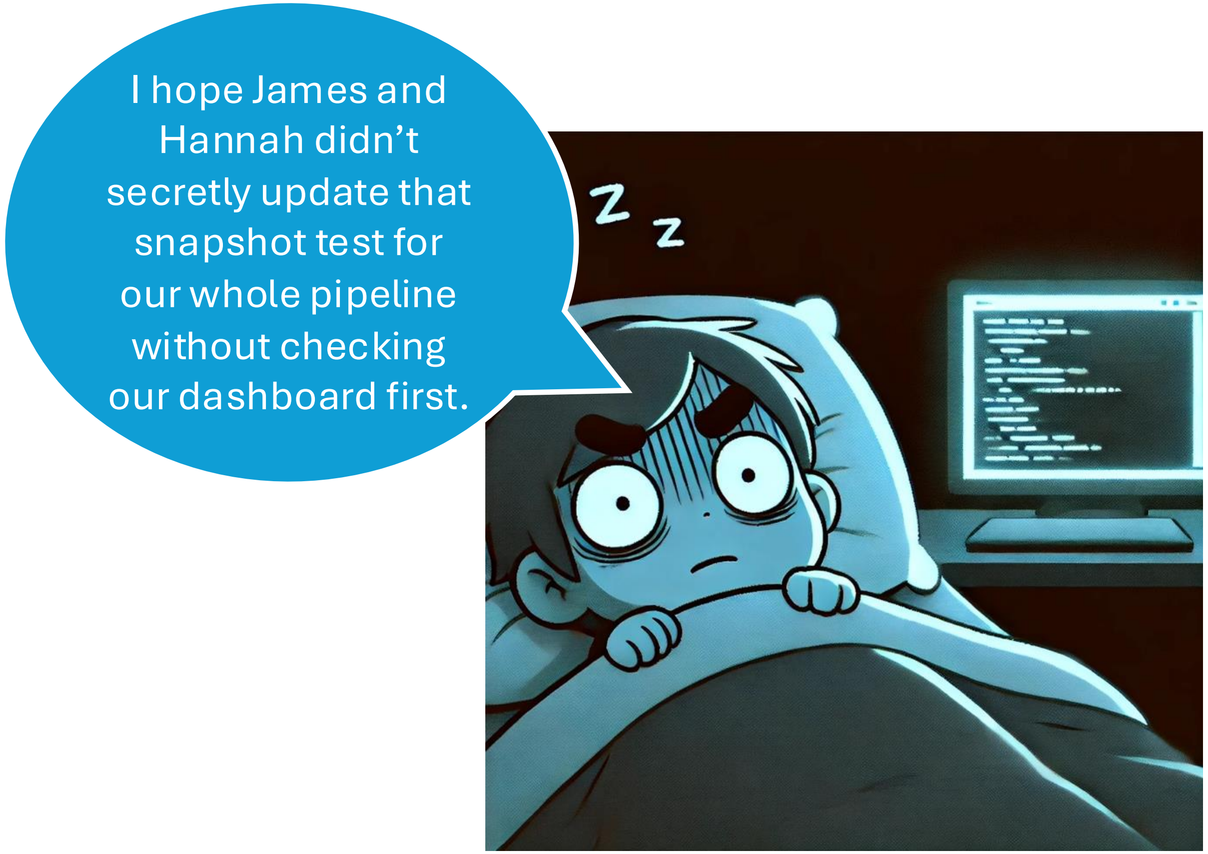
Does the UI operate as expected?

4. PROPERTY TESTING

Does it fit within the bounds of an expected output?

Snapshot Testing

- Probably not conceptually validated (using a previous out as a test case).
- **Fickle:** Needs to be changed often.
- Must be updated with each functional change.
- Verifies that only non-functional changes have occurred (i.e., refactors).
- Should only be updated with peer review.



Unit Testing

- Conceptually validates a function is performing correctly.
- Ideally, these are written for critical building blocks of a software.
- Helps developers sleep at night.

Test Driven Development (TDD)

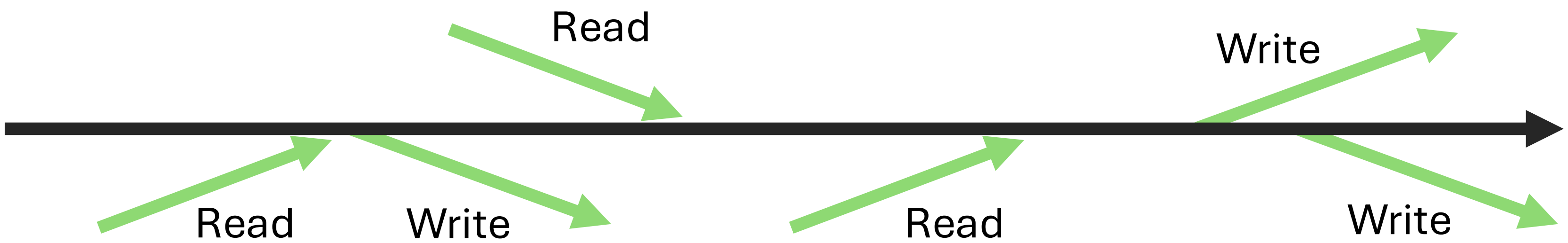
3 minutes ago (<1s) 6 Python

```python
# Explicitly run tests without unittest.main()
if __name__ == "__main__":
    # NOTE: This test case is expected to fail because the is_prime() function is not implemented
    suite = unittest.TestLoader().loadTestsFromTestCase(FailingTestPrimeChecker)
    unittest.TextTestRunner().run(suite)

    # NOTE: This test case is expected to pass
    suite = unittest.TestLoader().loadTestsFromTestCase(PassingTestPrimeChecker)
    unittest.TextTestRunner().run(suite)
```

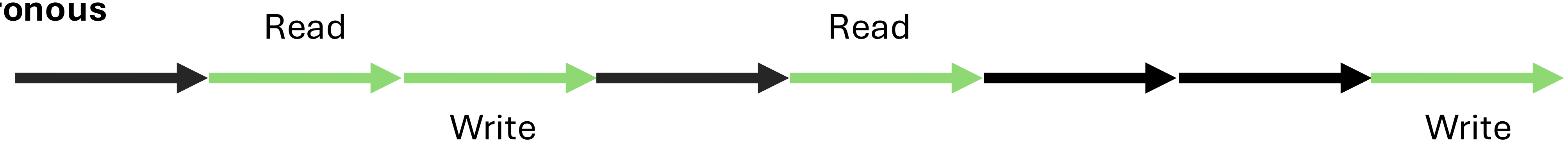
```
FAIL: test_prime_numbers (__main__.PassingTestPrimeChecker.test_prime_numbers)
```

Asynchronous

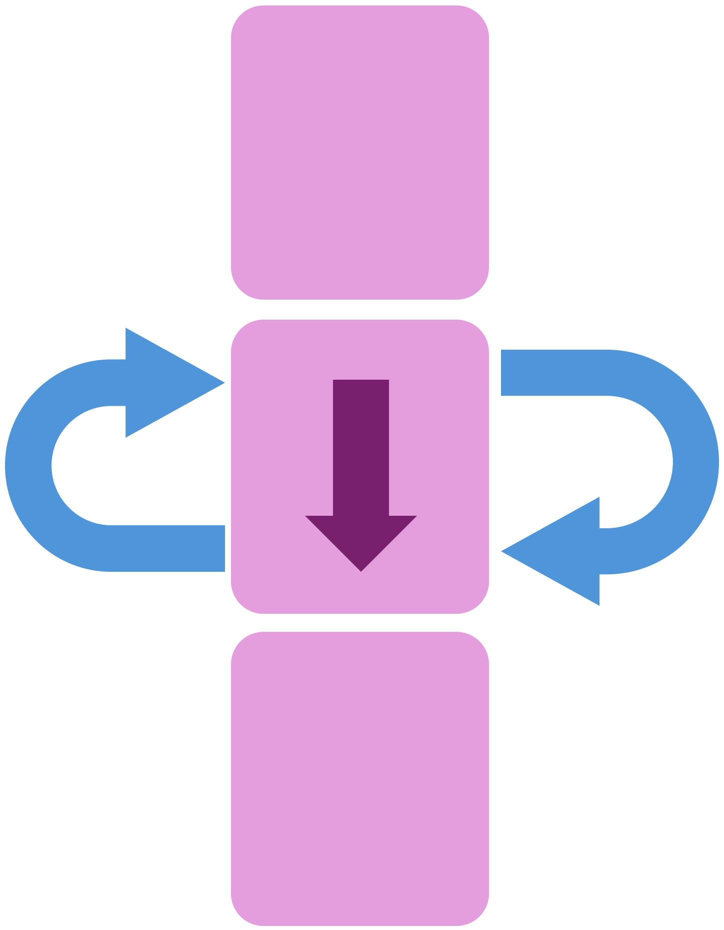


Asynchronous reads and writes run operations on separate threads decrease the runtime of the data pipeline.

Synchronous

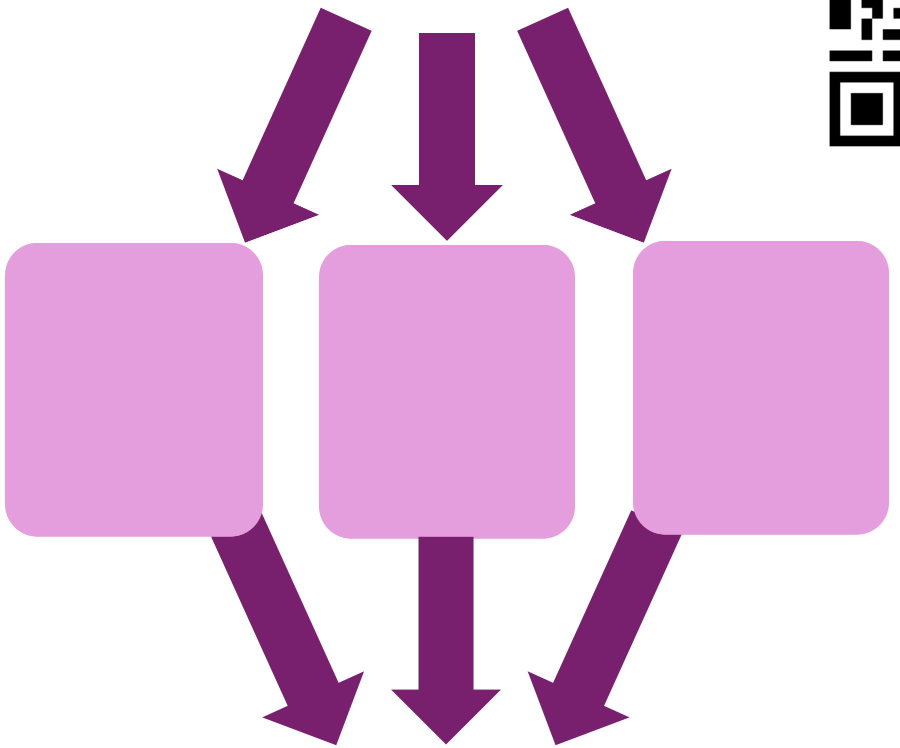


Chunking



Time Complexity: O(n)
Space Complexity: O(1)

Parallelization

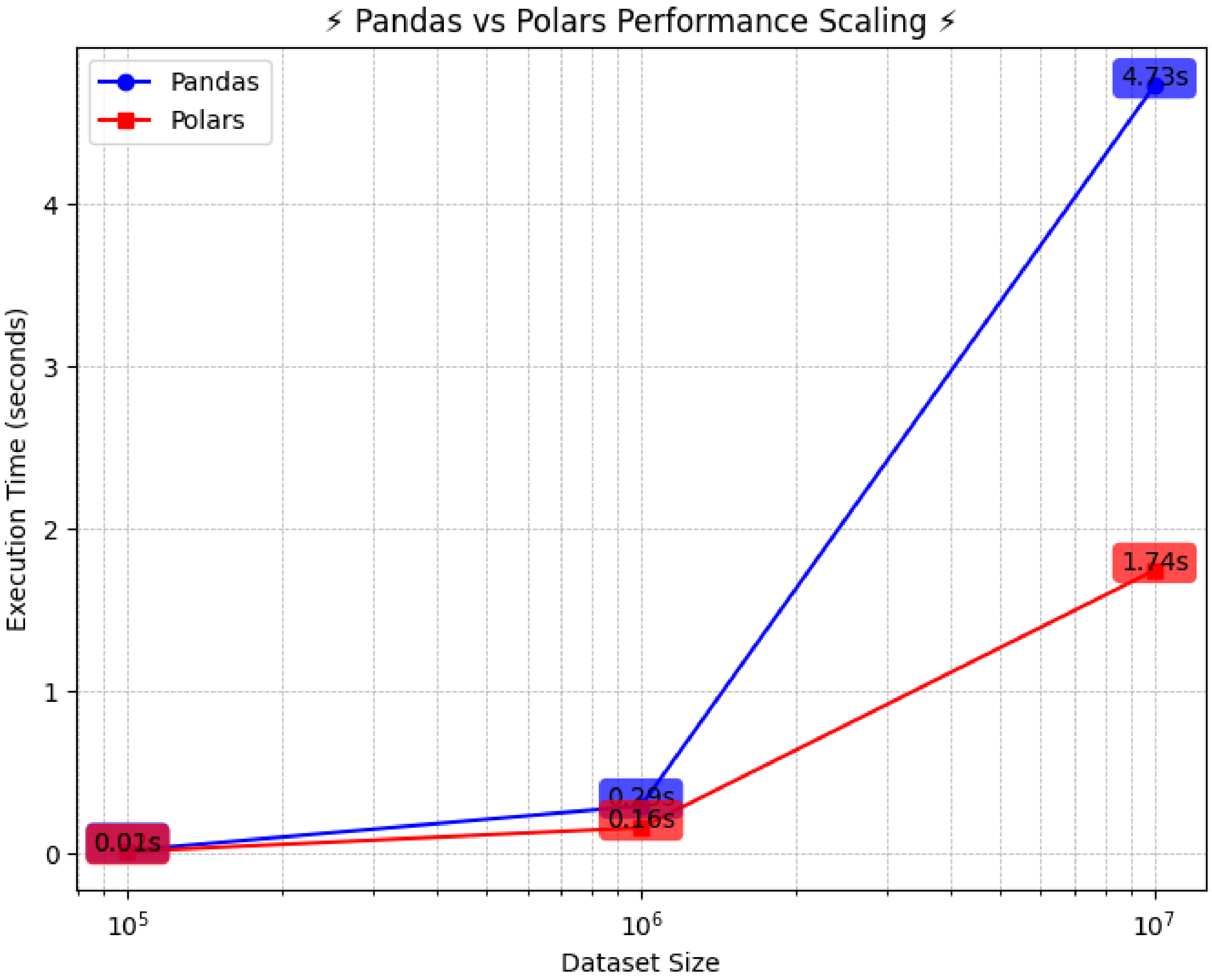


Time Complexity: O(1)
Space Complexity: O(n)

Programming Language Trade-Offs

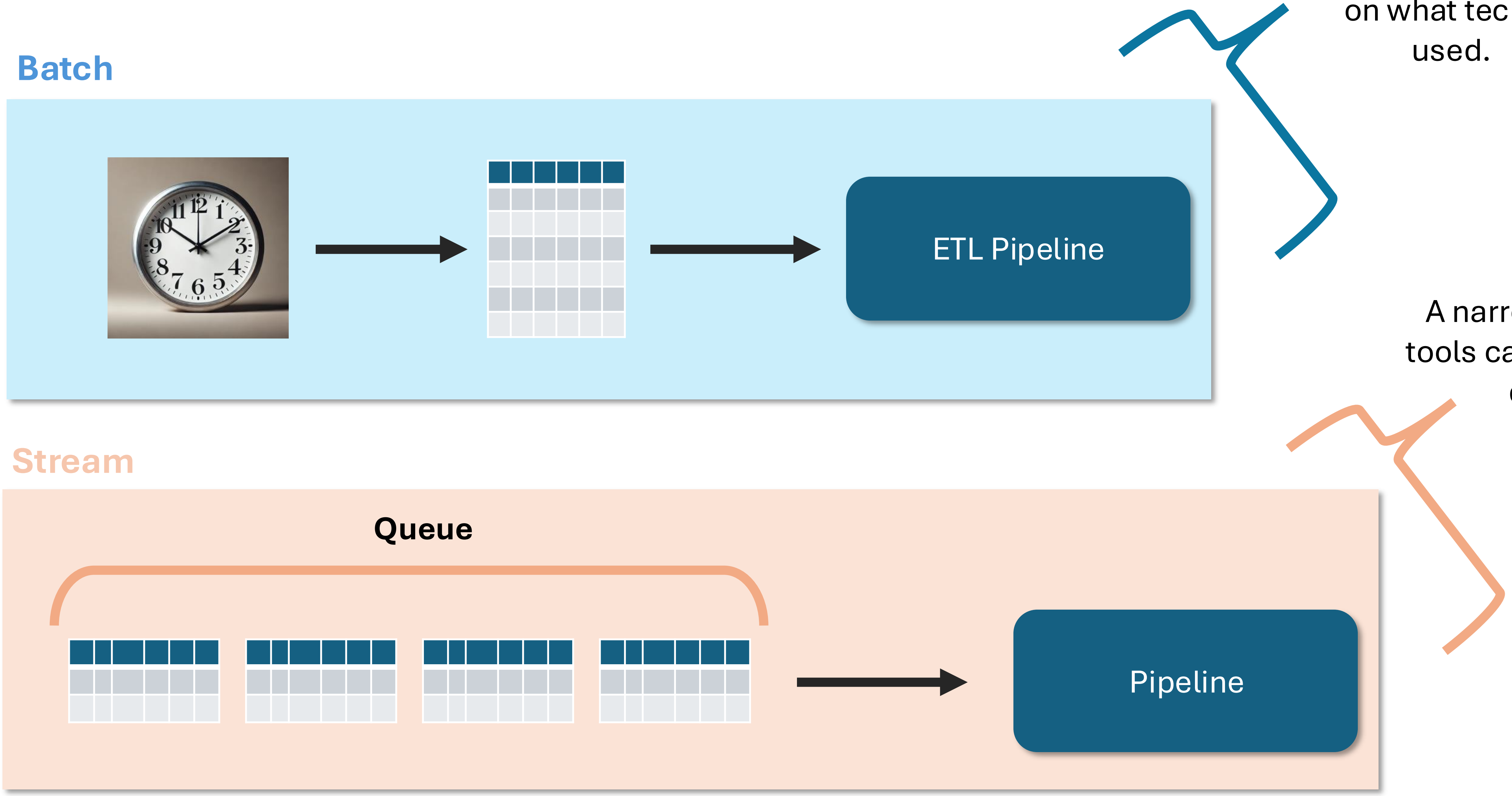
Use Case	Pandas	Polars	PySpark	SQL
Small datasets (<10M rows)	✅ Best	✅ Faster	❌ Overkill	✅ Good for querying
Large datasets (>10M rows, fits in RAM)	❌ Memory issues	✅ Efficient	✅ Good	✅ Good
Big Data (TB-scale, distributed)	❌ Impossible	❌ Limited	✅ Best	✅ Best
Parallel processing	❌ Single-threaded	✅ Multi-threaded	✅ Distributed	✅ Query optimizations
Complex ETL	✅ Simple	✅ Efficient	✅ Distributed pipelines	✅ SQL transformations
ML/Statistical modeling	✅ Best for ML	✅ Works	❌ Spark ML (limited)	❌ Not ideal
Cloud-based processing	❌ Local only	❌ Local only	✅ Cloud (Databricks, EMR)	✅ Cloud-native (BigQuery, Snowflake)

Pandas VS. Polars



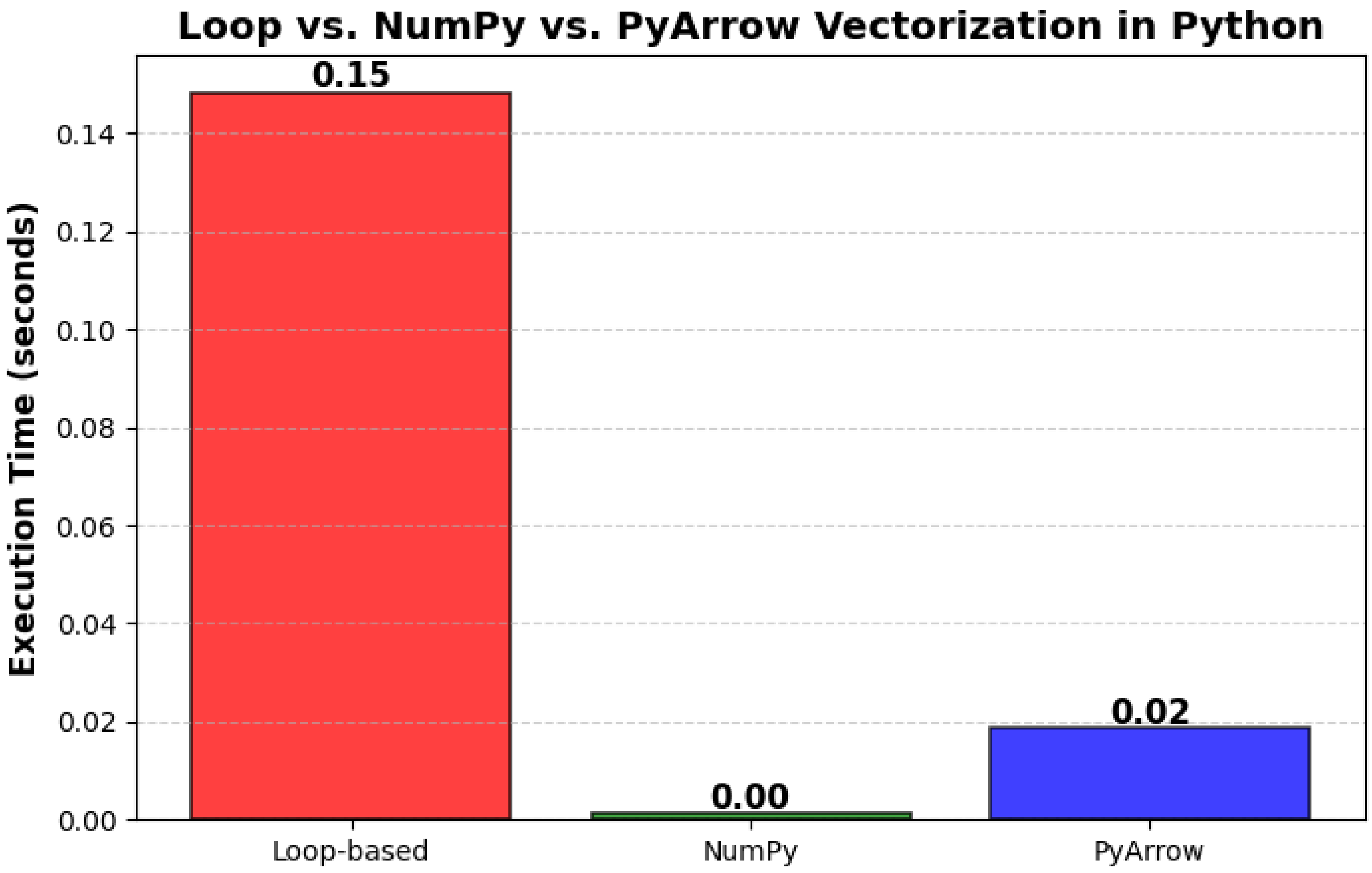
Batch or Stream Processing

Loose requirements on what tech is used.

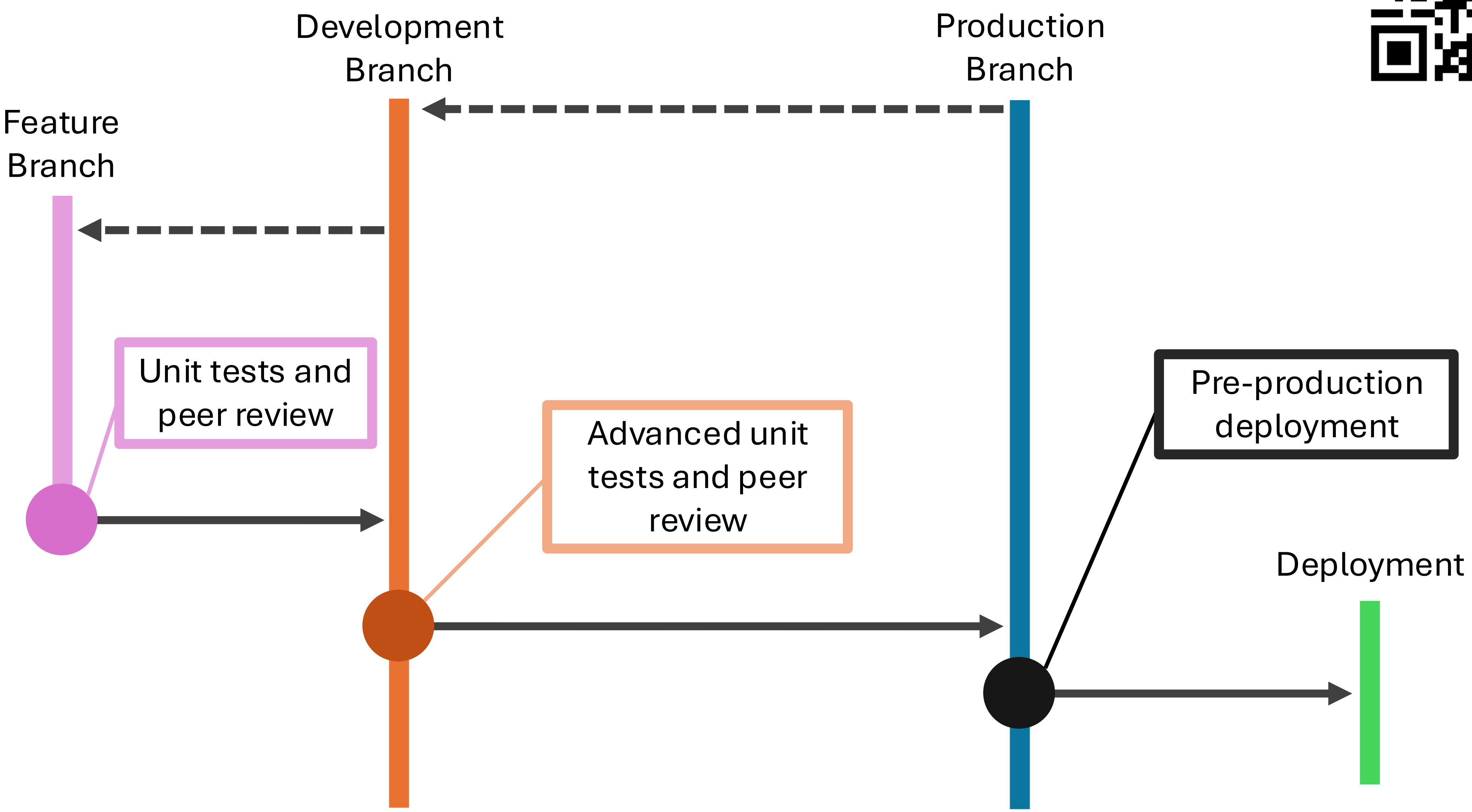


A narrow range of tools can get the job done.

Vectorization



Continuous Integration/Continuous Deployment
(CI/CD)



SQL Injection Attacks

```python
def unsafe_query():
    # Create a Snowpark session (Replace with actual credentials)
    connection_parameters = {
        "account": safe_account_retrieval_func(),
        "user": "your_user",
        "password": safe_password_retrieval_func(),
        "role": "your_role",
        "warehouse": "your_warehouse",
        "database": "your_database",
        "schema": "your_schema"
    }
    session = Session.builder.configs(connection_parameters).create()

    # 🚨 UNSAFE: Directly concatenating user input
    user_input = "1' OR '1'='1"  # An attacker injects this value
    query = f"SELECT * FROM users WHERE id = '{user_input}'"

    # Execute query
    df = session.sql(query).collect()  # This executes the malicious query
    print(df)
```

```python
def safe_query():
    # Create a Snowpark session (Replace with actual credentials)
    connection_parameters = {
        "account": safe_account_retrieval_func(),
        "user": "your_user",
        "password": safe_password_retrieval_func(),
        "role": "your_role",
        "warehouse": "your_warehouse",
        "database": "your_database",
        "schema": "your_schema"
    }
    session = Session.builder.configs(connection_parameters).create()

    # Secure version using bind parameters
    user_input = "1' OR '1'='1"  # Even if an attacker tries to inject, it won't work

    query = "SELECT * FROM users WHERE id = ?"
    df = session.sql(query).bind(user_input).collect()  # Secure query execution
    print(df)
```

INTERNET COMPUTER

Cloud Provider as State Machine Replication for Byzantine Fault Tolerance

INTERNET COMPUTER

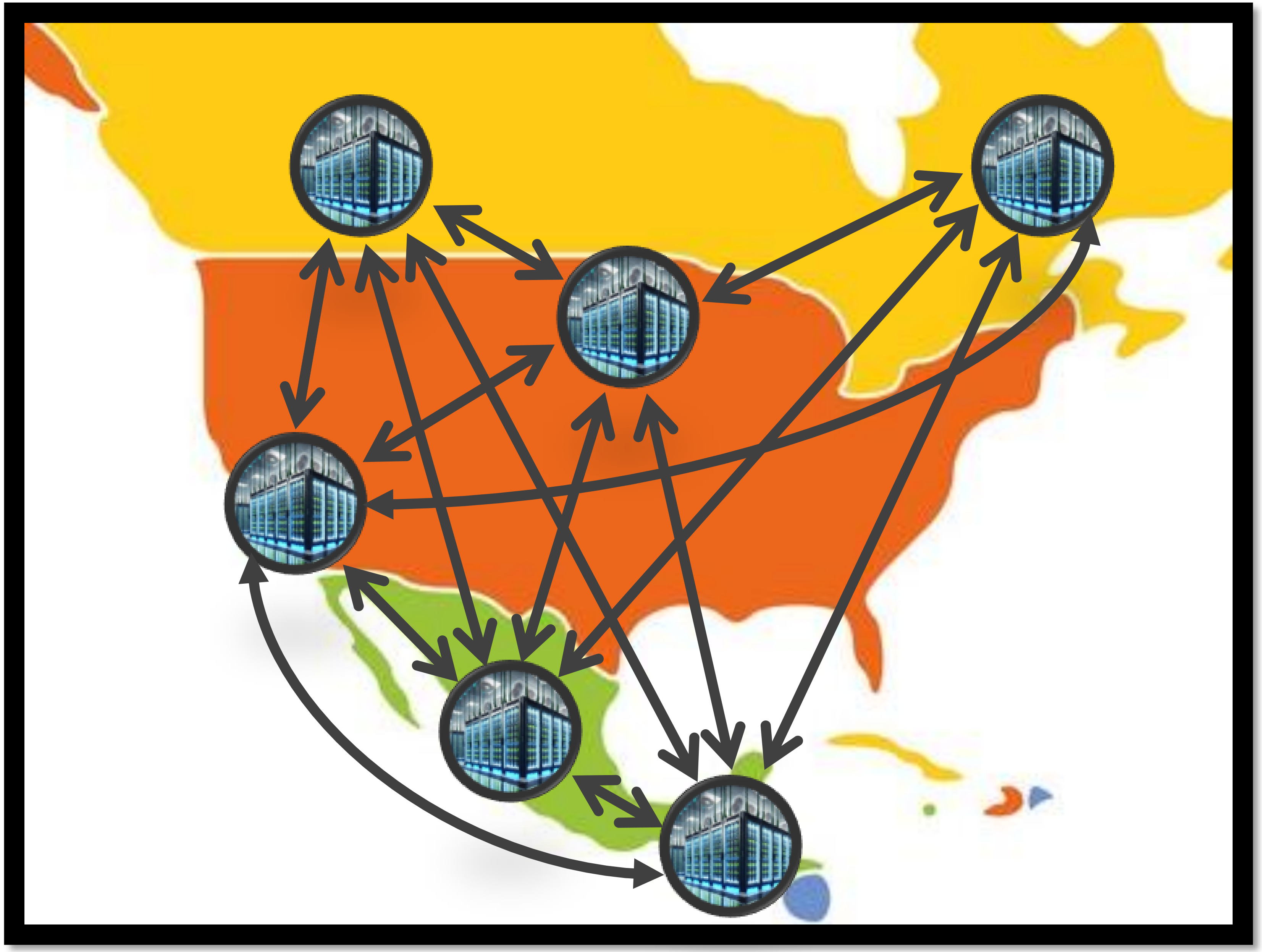
Subnets consist of 13 or more nodes. This allows for variable decentralization.

Latency is < 2 seconds.

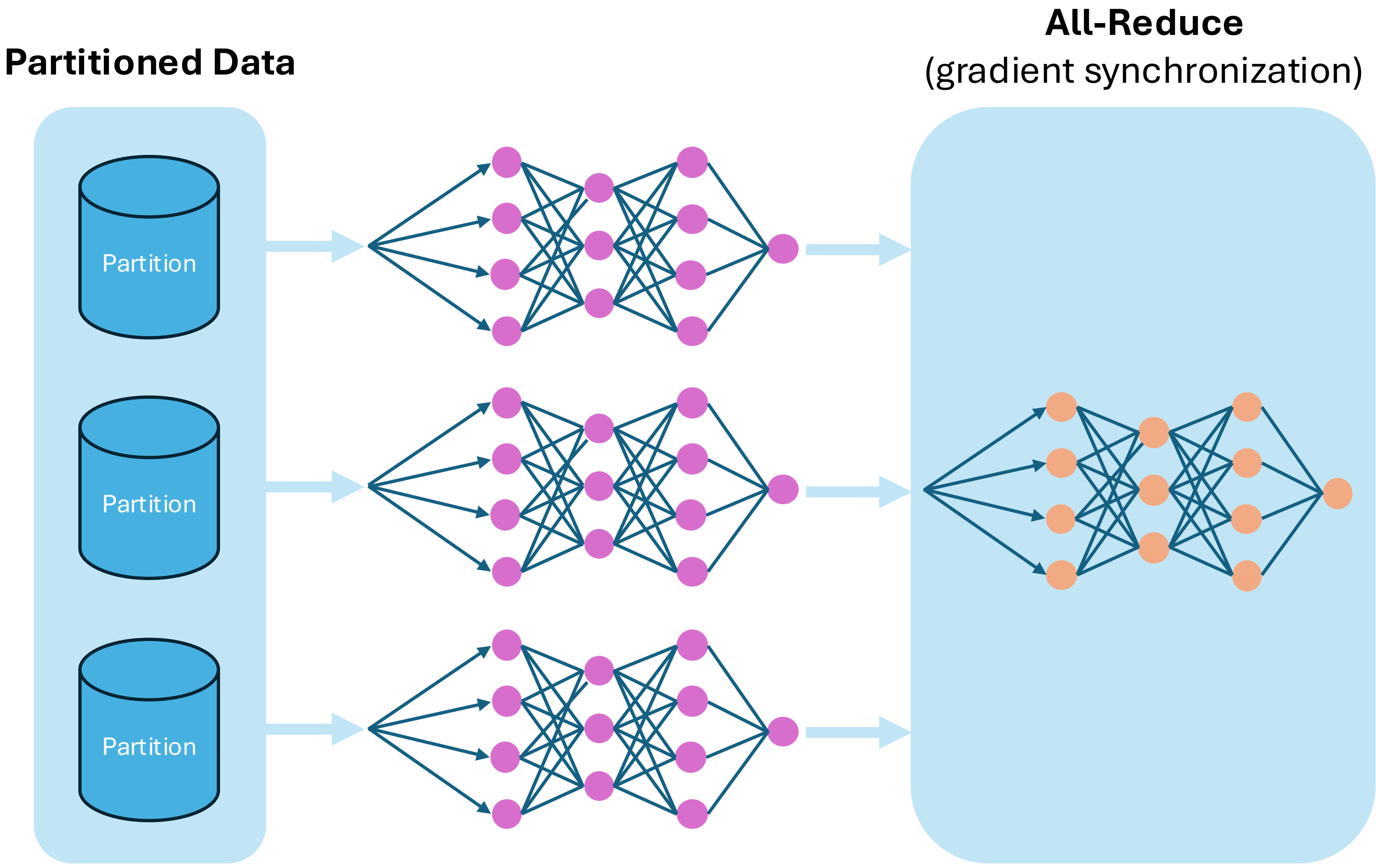
Egress (query calls) doesn't need to go through consensus.

Reverse gas: Developers pay for gas fees, and end-users can use the blockchain without needing to pay in ICP (like a normal website).

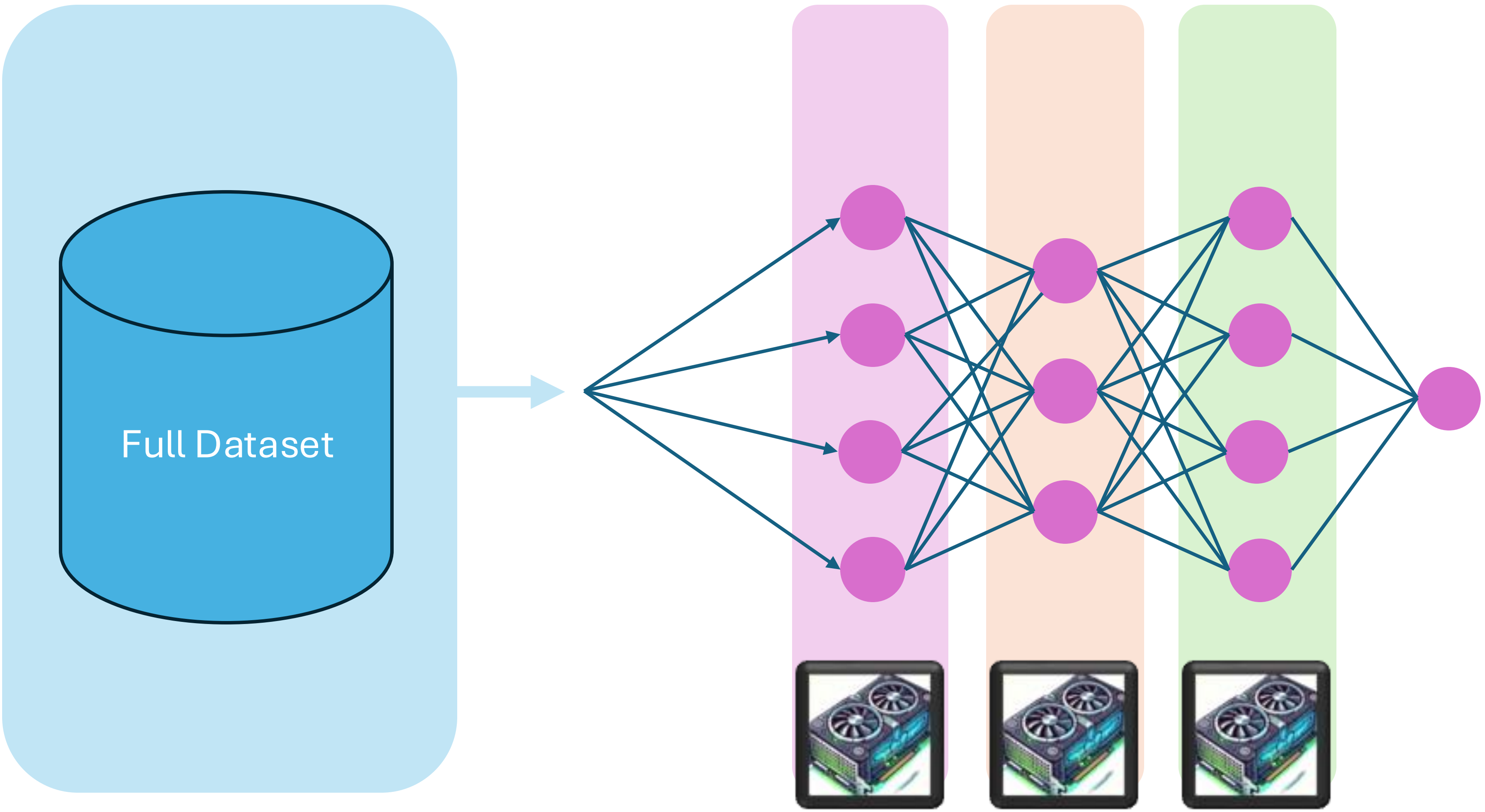
Software becomes immune to cyber hacks or datacenter outages due to any cause of failure.



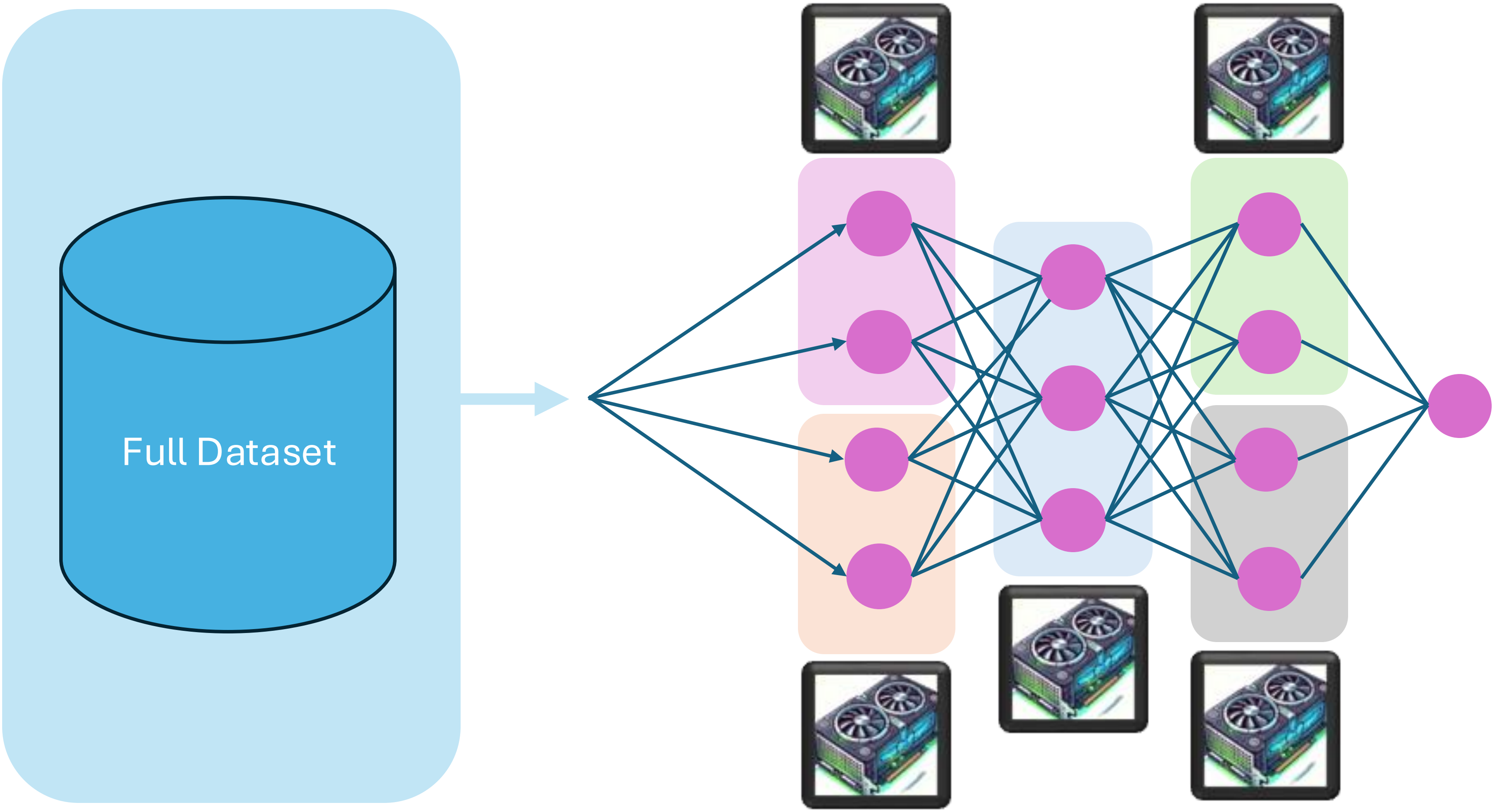
Distributed Data Parallel Training



Model Parallel Training



Tensor Parallel Training



Microsoft

ZeRO

Zero Redundancy Optimization

arXiv:1910.02054v3 [cs.LG] 13 May 2020

ZeRO: Memory Optimizations Toward Training Trillion Parameter Models

Samyam Rajbhandari*, Jeff Rasley*, Olatunji Ruwase, Yuxiong He
{samyamr, jerasley, olruwase, yuxhe}@microsoft.com

Abstract

Large deep learning models offer significant accuracy gains, but training billions to trillions of parameters is challenging. Existing solutions such as data and model parallelisms exhibit fundamental limitations to fit these models into limited device memory, while obtaining computation, communication and development efficiency. We develop a novel solution, Zero Redundancy Optimizer (*ZeRO*), to optimize memory, vastly improving training speed while increasing the model size that can be efficiently trained. *ZeRO* eliminates memory redundancies in data- and model-parallel training while retaining low communication volume and high computational granularity, allowing us to scale the model size proportional to the number of devices with sustained high efficiency. Our analysis on memory requirements and communication volume demonstrates: *ZeRO* has the potential to scale beyond 1 *Trillion* parameters using today's hardware.

We implement and evaluate ZeRO: it trains large models of over 100B parameter with super-linear speedup on 400 GPUs, achieving throughput of 15 Petaflops. This represents an 8x increase in model size and 10x increase in achievable performance over state-of-the-art. In terms of usability, *ZeRO* can train large models of up to 13B parameters (e.g., larger than Megatron GPT 8.3B and T5 11B) without requiring model parallelism which is harder for scientists to apply. Last but not the least, researchers have used the system breakthroughs of *ZeRO* to create the world's largest language model (17B parameters) with record breaking accuracy.

1 Extended Introduction

Deep Learning (DL) models are becoming larger, and the increase in model size offers significant accuracy gain. In the area of Natural Language Processing (NLP), the transformers have paved way for large models like Bert-large (0.3B) [1], GPT-2 (1.5B) [2], Megatron-LM (8.3B) [3], T5 (11B) [4]. To enable the continuation of model size growth from 10s of billions to trillions of parameters, we experience the challenges of training them - they clearly do not fit within the memory of a single device, e.g., GPU or TPU, and simply adding more devices will not help scale the training.

Basic data parallelism (DP) does not reduce memory per device, and runs out of memory for models with more than 1.4B parameters on current generation of GPUs with 32 GB memory. Other existing solutions such as Pipeline Parallelism (PP), Model Parallelism (MP),

*Equal Contributors

Microsoft

Graviton

Trusted Execution Environments (TEEs) on GPUs

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Graviton: Trusted Execution Environments on GPUs

Stavros Volos and Kapil Vaswani, *Microsoft Research;*
Rodrigo Bruno, *INESC-ID / IST, University of Lisbon*

https://www.usenix.org/conference/osdi18/presentation/volos

This paper is included in the Proceedings of the
13th USENIX Symposium on Operating Systems Design
and Implementation (OSDI '18).
October 8–10, 2018 • Carlsbad, CA, USA
ISBN 978-1-939133-08-3

Open access to the Proceedings of the
13th USENIX Symposium on Operating Systems
Design and Implementation
is sponsored by USENIX.

Characterization of GPU TEE Overheads in Distributed Data Parallel ML Training

Jonghyun Lee
Department of Electrical Computer Engineering.
University of Southern California
Los Angeles, USA
leejongh@usc.edu

Yongqin Wang
Department of Electrical Computer Engineering.
University of Southern California)
Los Angeles, USA
yongqin@usc.edu

Rachit Rajat
Department of Electrical Computer Engineering.
University of Southern California
Los Angeles, USA
rrajat@usc.edu

Murali Annavaram
Department of Electrical Computer Engineering.
University of Southern California
Los Angeles, USA
annavara@usc.edu

arXiv:2501.11771v1 [cs.CR] 20 Jan 2025

***Abstract*—Confidential computing (CC) or trusted execution enclaves (TEEs) is now the most common approach to enable secure computing in the cloud. The recent introduction of GPU TEEs by NVIDIA enables machine learning (ML) models to be trained without leaking model weights or data to the cloud provider. However, the potential performance implications of using GPU TEEs for ML training are not well characterized. In this work, we present an in-depth characterization study on performance overhead associated with running distributed data parallel (DDP) ML training with GPU Trusted Execution Environments (TEE).**

Our study reveals the performance challenges in DDP training within GPU TEEs. DDP uses ring-all-reduce, a well-known approach, to aggregate gradients from multiple devices. Ring all-reduce consists of multiple scatter-reduce and all-gather operations. In GPU TEEs only the GPU package (GPU and HBM memory) is trusted. Hence, any data communicated outside the GPU packages must be encrypted and authenticated for confidentiality and integrity verification. Hence, each phase of the ring-all-reduce requires encryption and message authentication code (MAC) generation from the sender, and decryption and MAC authentication on the receiver. As the number of GPUs participating in DDP increases, the overhead of secure inter-GPU communication during ring-all-reduce grows proportionally. Additionally, larger models lead to more asynchronous all-reduce operations, exacerbating the communication cost. Our results show that with four GPU TEEs, depending on the model that is being trained, the runtime per training iteration increases by an average of 8x and up to a maximum of 41.6x compared to DDP training without TEE.

***Index Terms*—Trusted Execution Environment, GPU TEE, Distributed Data Parallel, multi-GPU training.**

I. Introduction

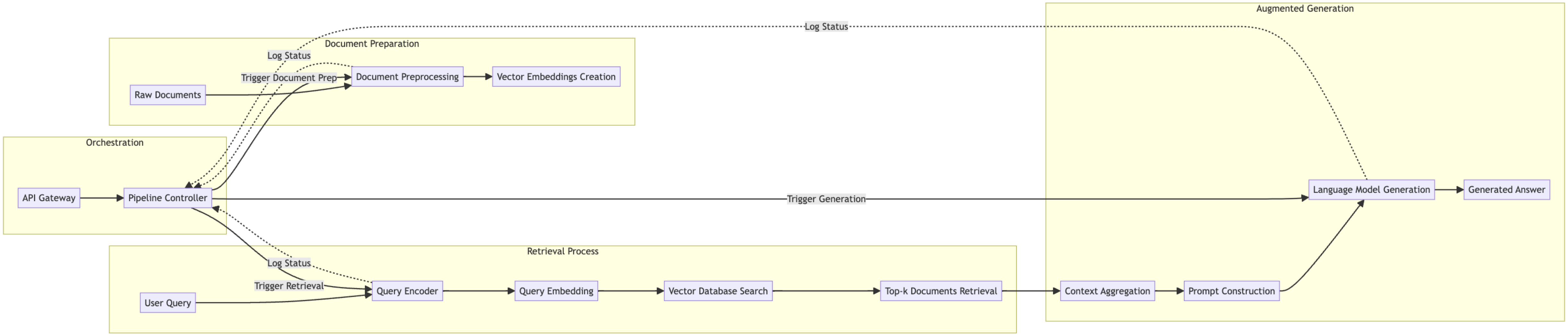
In recent years, machine learning (ML) providers have increasingly relied on cloud computing platforms to deploy large-scale ML workloads on multi-GPU machines for their scalability and cost-effectiveness. However, running ML training on these cloud servers introduces vulnerabilities to proprietary or sensitive data and models, including risks from physical attacks on the underlying hardware and compromised host operating systems. To address these security concerns, various hardware-enforced Trusted Execution Environments (TEEs) have been introduced. Intel Software Guard Extensions (SGX), ARM TrustZone, AMD Secure Encrypted Virtualization (SEV), and Intel Trusted Domain Extensions (TDX). These TEEs enforce isolation by rejecting virtual memory translation requests of unauthorized entities to the guest machine's memory. Additionally, DRAM encryption and authentication safeguards against potential hardware attacks such as memory bus probing and data replay attacks. Since modern ML workloads prefer to exploit massive parallelism in GPUs, recently, GPUs also started supporting TEEs for private ML training. Notably, solutions such as Graviton [27], HIX [14], and NVIDIA Confidential Computing (CC) [23] extend TEE capabilities to commodity GPUs. NVIDIA H100 GPUs introduced the first commercially enabled GPU TEE.

NVIDIA CC has demonstrated minimal overhead when data transmission to and from GPU TEEs is limited [23]. However, modern ML models are often trained using multiple GPUs in the distributed data parallel (DDP) approach where each GPU receives a fraction of the total training data. The ML model itself learns from all the data partitions by frequently synchronizing gradients from all the GPUs and updating the model parameters from the gradients averaged over all the partitions. In the context of secure training using multiple GPU TEEs, this synchronization involves encrypting/decrypting and authenticating gradients before aggregation. Despite the importance of TEE-enabled private training in DDP, the overhead associated with gradient synchronization in GPU TEEs remains underexplored in the existing literature. In this work, we analyze the overhead of private DDP training using multiple NVIDIA H100 GPU TEEs.

System Topology: Figure 1 illustrates the general multi-GPU system topology for secure ML training in NVIDIA CC. The system consists of a CPU TEE and GPU TEEs connected via PCI-e and NVLink. CPU TEE, implemented using an

RAG

Retrieval Augmented Generation



Questions?