MONITORING, EVALUATION & SAFETY

LLM Evaluation
Benchmark, score, and compare LLM outputs across tasks and safety metrics

`LM Eval Harness` `HELM` `Ragas`

Drift & Monitoring
Detect data drift, concept drift, and model degradation in production

`Evidently` `WhyLabs` `Fiddler`

Observability & Tracing
Trace LLM calls, agent steps, latency, and costs across the stack

`LangSmith` `Arize` `Helicone`

Guardrails & Safety
Content filters, output validation, toxicity detection, policy enforcement

`Guardrails AI` `NeMo Guardr.`

SERVING, DEPLOYMENT & INFERENCE

LLM Serving
High-throughput, low-latency serving engines for large language models

`vLLM` `TGI` `TensorRT-LLM` `SGLang`

Model Serving
Deploy, version, and serve ML models behind production APIs

`TorchServe` `Triton` `BentoML` `Seldon`

MLOps & Registries
Model versioning, CI/CD, packaging, and reproducible deployment pipelines

`MLflow` `SageMaker` `Vertex AI` `Kubeflow`

Compression & Edge
Quantization, pruning, distillation for on-device and low-latency deployment

`GPTQ` `AWQ` `llama.cpp` `ONNX`

LLM & GENERATIVE AI INFRASTRUCTURE

Foundation Model APIs
Access to frontier LLMs via API for reasoning, generation, and dialog

`OpenAI` `Anthropic` `Google` `Cohere`

Open Model Hubs
Download, share, and deploy open-weight models and datasets

`HuggingFace` `Ollama` `Replicate`

RAG & Retrieval
Augment LLMs with external knowledge via retrieval pipelines and chunking

`LlamaIndex` `LangChain` `Haystack`

Fine-Tuning & Alignment
Adapt foundation models via RLHF, LoRA, prompt tuning, and distillation

`TRL` `PEFT` `DSPy` `Axolotl`

In practice, building an AI solution means selecting and manually stitching together >>1 AI systems from a massive set of tools into a working whole

AGENT FRAMEWORKS & ORCHESTRATION

Agent Frameworks
Build autonomous agents that reason, plan, use tools, and take actions

`LangGraph` `CrewAI` `AutoGen`

Multi-Agent Systems
Coordinate teams of agents with roles, delegation, and consensus protocols

`CrewAI` `AutoGen` `MetaGPT`

Tool & API Integration
Connect agents to external services, functions, databases, and code execution

`MCP` `Function Calling`

Workflow Orchestration
DAG-based pipelines, conditional routing, and scheduling of AI workflows

`Prefect` `Dagster` `Temporal`

MONITORING, EVALUATION & SAFETY

LLM Evaluation
Benchmark, score, and compare LLM outputs across tasks and safety metrics

`LM Eval Harness` `HELM` `Ragas`

Drift & Monitoring
Detect data drift, concept drift, and model degradation in production

`Evidently` `WhyLabs` `Fiddler`

Observability & Tracing
Trace LLM calls, agent steps, latency, and costs across the stack

`LangSmith` `Arize` `Helicone`

Guardrails & Safety
Content filters, output validation, toxicity detection, policy enforcement

`Guardrails AI` `NeMo Guardr.`

THERE ARE THREE CRITICAL FEATURES IN AI DEVELOPMENT
SPEED, SPEED, & SPEED

Time to market, Better models, More models

~90%

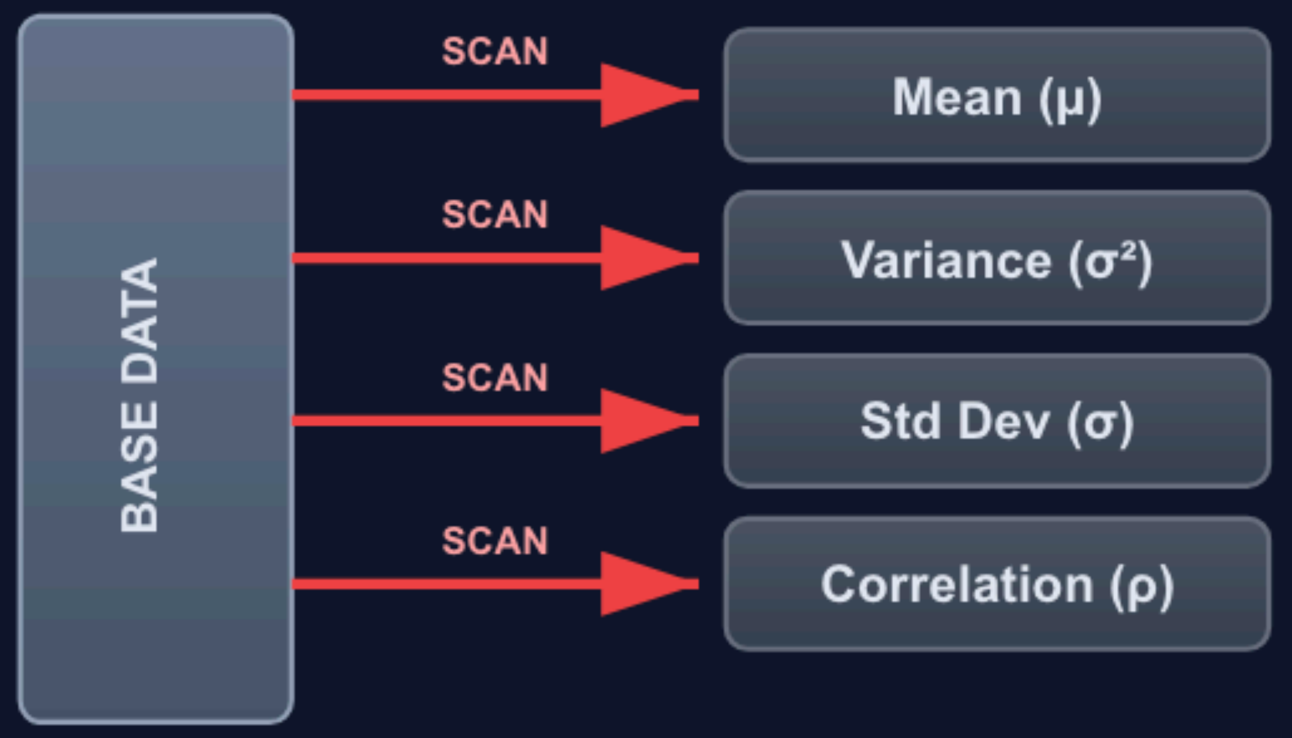
OF EFFORT IN AI GOES INTO “GLUE ENGINEERING”

OF LATENCY AND $$$ IN AI IS BECAUSE OF STORAGE

Data Canopy: Storage-Aware Caching for Statistical Analysis

Decompose statistics into reusable basic aggregates → Avoid redundant data movement

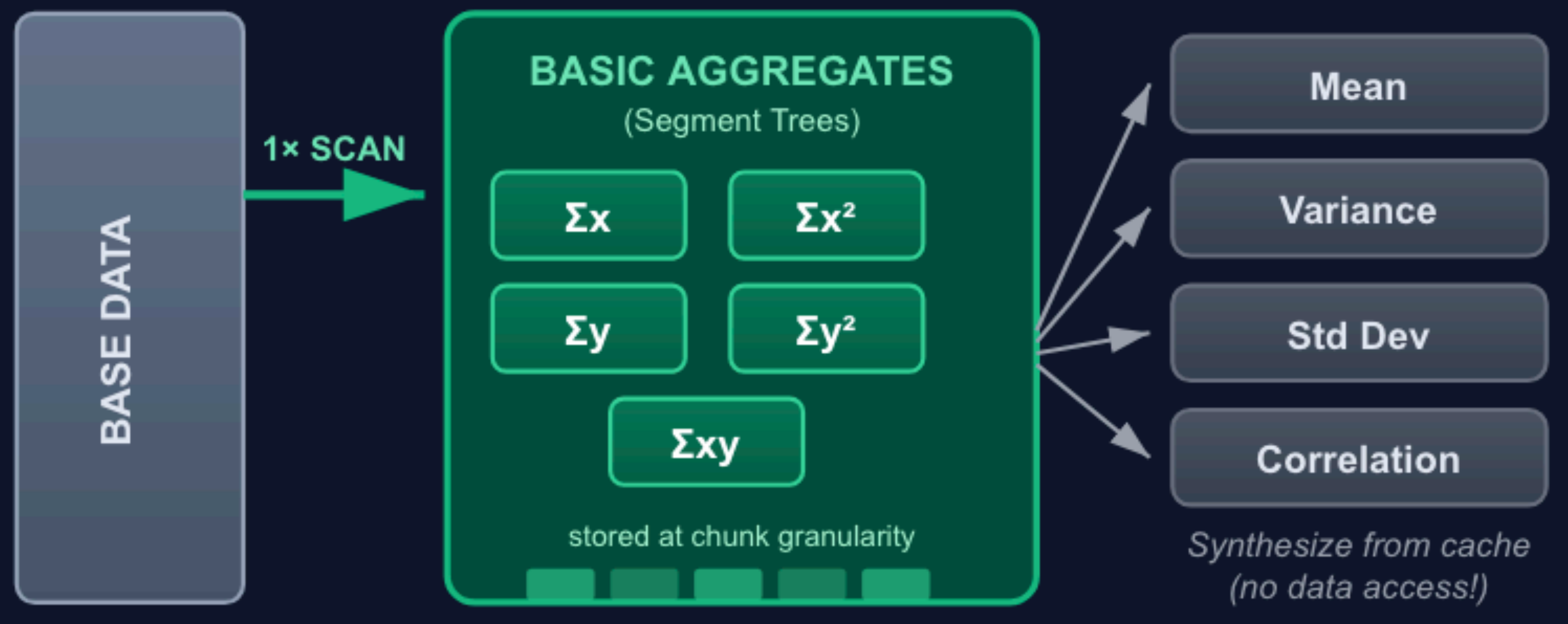
Traditional: Repeated Full Scans



4× Full Data Movement → Slow

Each query re-reads entire dataset

Data Canopy: Cache Basic Aggregates



1× Data Movement → Fast

Future queries synthesize from cached aggregates

Statistics Share Basic Aggregates

Statistic	Σx	Σx^2	Σxy	Σy^2	Σy
Mean	●				
Root Mean Sq		●			
Variance	●	●			
Std Deviation	●	●			
Kurtosis	●	●			
Covariance	●		●		●
Linear Regr.	●	●	●	●	●
Correlation	●	●	●	●	●

● = needed

90%+ of NumPy/SciPy statistics

Impact on ML Algorithms

Linear Regression	10^6× faster
Bayesian Classification	10^3× faster
Collaborative Filtering	10^6× faster

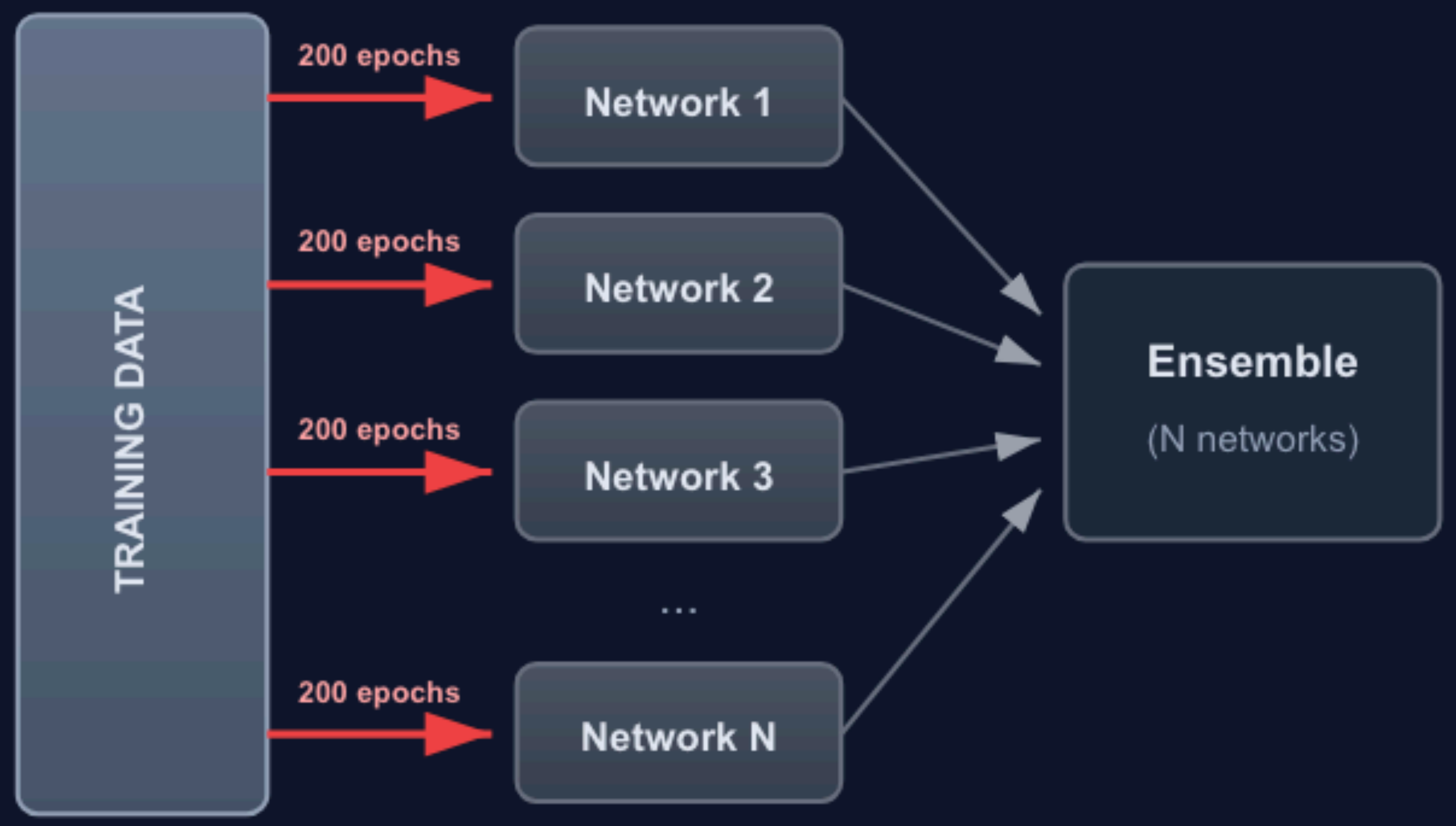
Performance Improves With Use

More queries → more aggregates cached → faster future queries
10× speedup after 100 queries

MotherNets: Rapid Deep Ensemble Learning

Share training computation across ensemble networks → Amortize data movement costs

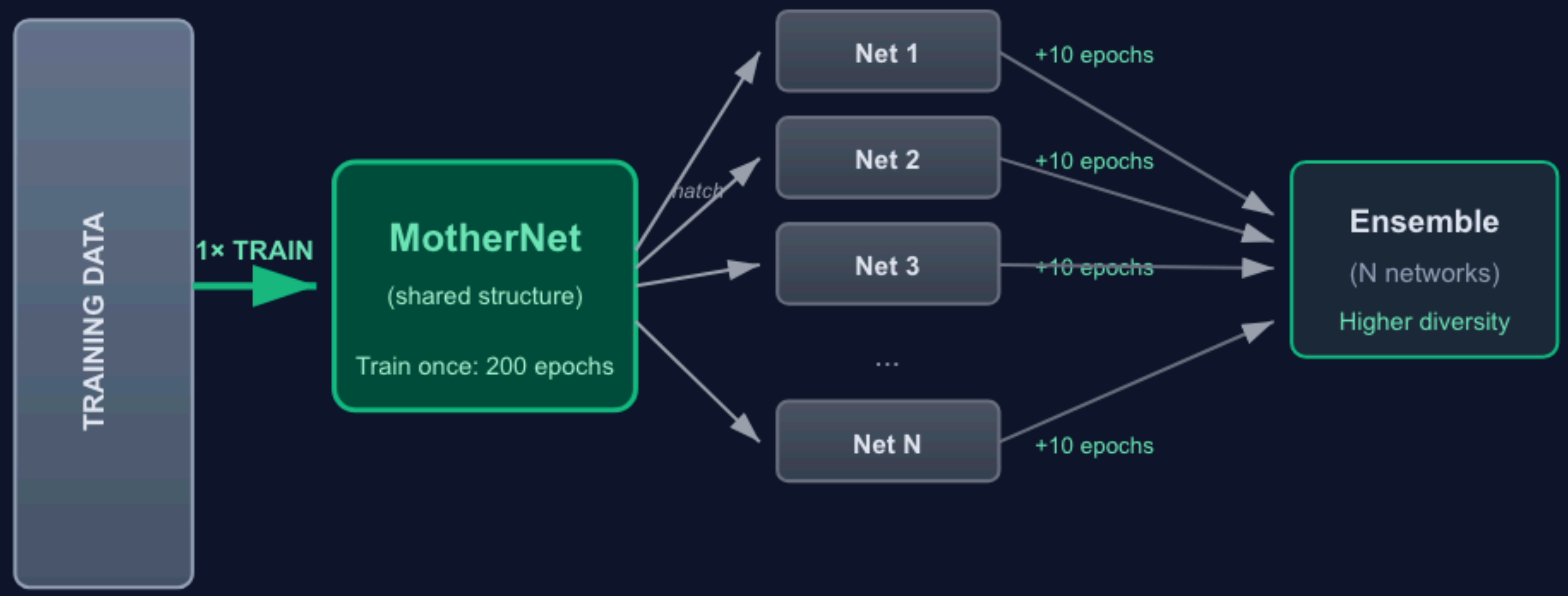
Traditional: Train Each Network From Scratch



N × Full Training Cost → Linear Growth

Each network re-processes entire dataset from scratch

MotherNets: Share Training via Structural Similarity



1× Full + N× Quick Training → Sub-linear Cost

"Share epochs" - train common structure once, specialize quickly

Ensemble Training Methods Comparison

Method	Fast	High Acc	Diverse	Large Size
Full Data	✗	●	✗	✗
Bagging	~	✗	✗	✗
Knowledge Dist.	~	✗	✗	✗
Snapshot Ens.	●	~	✗	✗
MotherNets	●	●	●	●

● = yes
✗ = no
~ = partial

New Pareto frontier

Results: Better Accuracy AND Speed

vs Snapshot Ensembles: **35% faster**

vs Knowledge Distill: **2-4× faster**

Test error improvement: **2-3% better**

Scales to Large Ensembles

More networks → more shared computation
100 networks: 10+ hours saved vs Snapshot


μ-TWO: 3× Faster Multi-Model Training
Overlap data movement with independent compute → Maximize GPU utilization
Traditional: Fuse All or Train Sequentially
Option A: Complete Fusion
All 8 Models Fused into Single Graph
Memory: 6× GPU capacity! → OOM
With Swapping:
FW+BW (fused)
STALL
ops
STALL
ops
swap
swap
No independent ops → can't hide swaps!
Option B: Train Sequentially
M1-M2
M3-M4
M5-M6
M7-M8
GPU underutilized (~50%)
GPU Utilization
~50%
Memory Limited OR GPU Cycles Wasted
Feature maps sit idle between forward and backward passes
μ-TWO: Sub-Array Fusion + Smart Scheduling
Sub-Array A (M1-M4)
Fused: FW_A | BW_A
Independent graphs
Sub-Array B (M5-M8)
Fused: FW_B | BW_B
Independent graphs
Key Insight:
BW_A ⊥ FW_B
Multiplexed Schedule: Overlap Swaps with Compute
Compute:
BW_A
FW_B
BW_A
FW_B
BW_A
FW_B
Swap:
prefetch
prefetch
prefetch
Swaps hidden behind FW_B compute → Zero stalls!
GPU Utilization
~95%
Saturate Compute + Hide Memory Latency
Multiplex BW of one sub-array with FW of another
The Trade-off Space
Compute Utilization
Complete Fusion
μ-TWO
No Fusion
Peak Memory
Independent Ops
Approach Comparison
Complete Fusion: High compute, but OOM / many stalls
No Fusion: Low memory, but compute underutilized
μ-TWO: Best of both worlds!
Results: Faster Training, More Models
Training speedup:
Up to 3×
More models per GPU:
3-5× more
Memory footprint:
Up to 6× GPU mem
GPU hours saved:
5-40 hours
Scales Across Diverse Models
Vision
ViT, ResNet
NLP
BERT, GPT2
RecSys
DLRM
More models → more overlap
Speedup scales with # of models!
Enables: Hyperparameter tuning • Ensemble learning • Neural Architecture Search