XGBoost

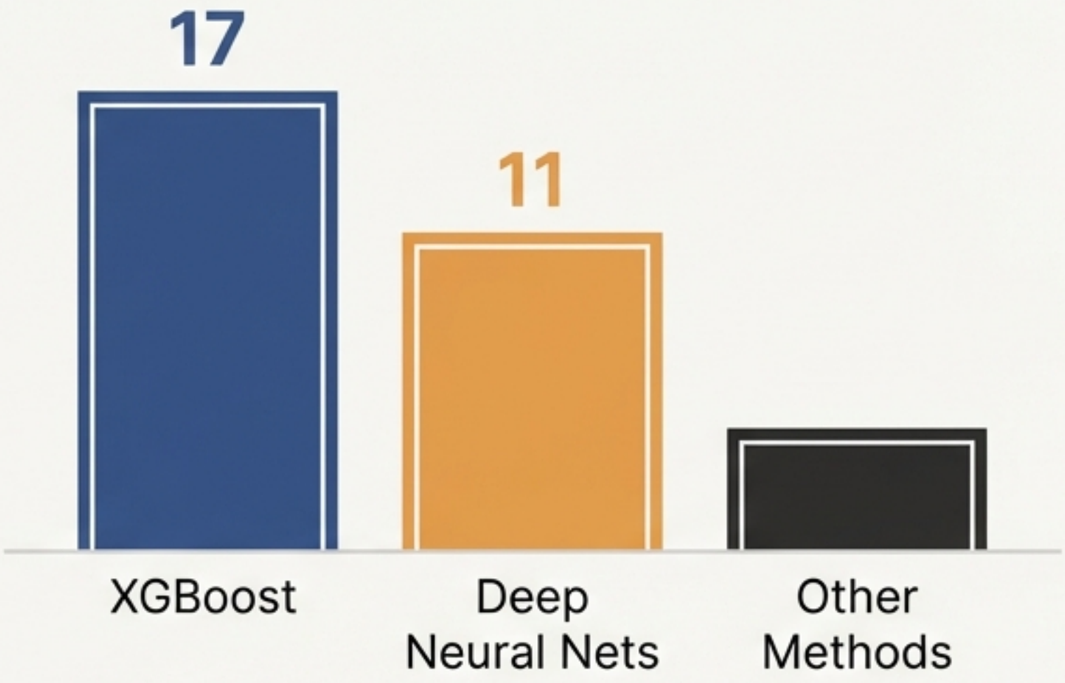
XGBoost: The Anatomy of a Champion

A deep dive into the algorithmic and system innovations that created the dominant model in machine learning competitions.

NotebookLM

XGBoost is the consensus choice of winners.

Among the **29** challenge winning solutions published at Kaggle's blog during 2015, **17 solutions used XGBoost**.



- In the 2015 KDDCup, **every single winning team** in the top 10 used XGBoost.
- Winning applications span a huge range of problems: sales prediction, web text classification, ad click-through rate prediction, and high-energy physics.

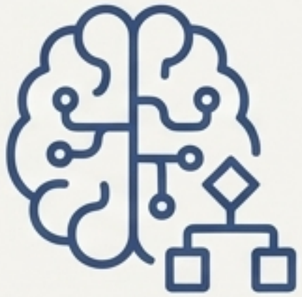
This dominance is not accidental. ***What is XGBoost's secret sauce?***

NotebookLM

Its success lies in the co-design of a smarter algorithm and a faster system.

Part 1: The Smarter Algorithm

Precision and Generalisation



- A regularised objective to control for overfitting.
- Novel approaches for handling real-world sparse data.
- A theoretically-sound method for approximating splits on massive datasets.

Part 2: The Blazing-Fast System

Engineering for Speed and Scale

- A purpose-built data structure for parallel learning.
- Cache-aware algorithms that minimise hardware bottlenecks.
- Out-of-core computation to process data that exceeds RAM.

NotebookLM

The algorithm's foundation is a regularised objective that prevents overfitting.

Standard gradient boosting optimises a loss function. XGBoost improves this by adding a regularisation term that penalises model complexity. This helps to smooth the final learnt weights and create simpler, more generalisable models.

$$L(\phi) = \sum l(\hat{y}_i, y_i) + \sum \Omega(f_k)$$

Training Loss
Measures how well the model fits the training data.
(e.g., Logistic Loss, Square Error)

Regularisation Term
Penalises model complexity to prevent overfitting.

$$\Omega(f) = \gamma T + \frac{1}{2}\lambda \|\boldsymbol{w}\|^2$$

Controls the number of leaves.

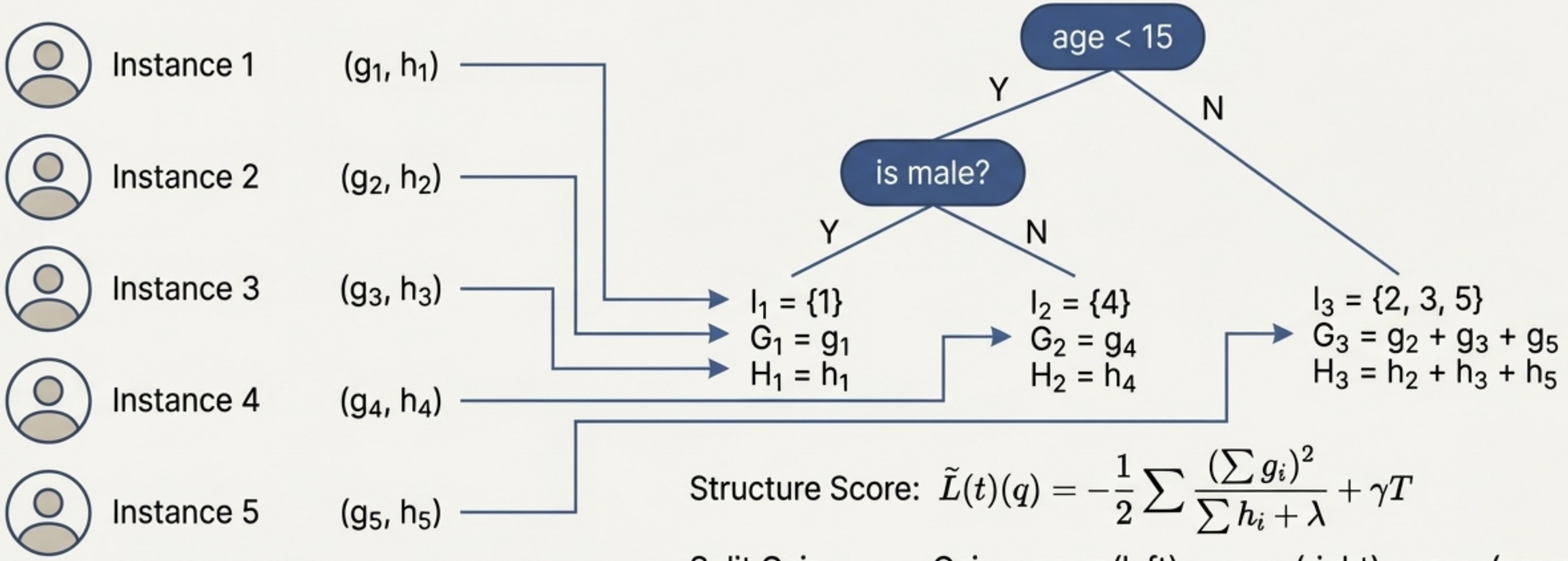
Controls the magnitude of leaf weights (L2 regularisation).

Key Takeaway: By balancing predictive accuracy with model simplicity, XGBoost builds models that are less likely to overfit the training data.

NotebookLM

A unique quality score guides the search for the optimal tree structure.

At each step, the algorithm greedily adds the branch that provides the biggest improvement. This improvement is measured by a “structure score” derived from the first and second-order gradient statistics of the instances in each leaf.



Structure Score: $\tilde{L}(t)(q) = -\frac{1}{2}\sum \frac{(\sum g_i)^2}{\sum h_i + \lambda} + \gamma T$

Split Gain: Gain = score(left) + score(right) - score(parent)

Key Takeaway: This principled **scoring function** allows the model to evaluate the quality of any potential tree structure against the regularised objective.

NotebookLM

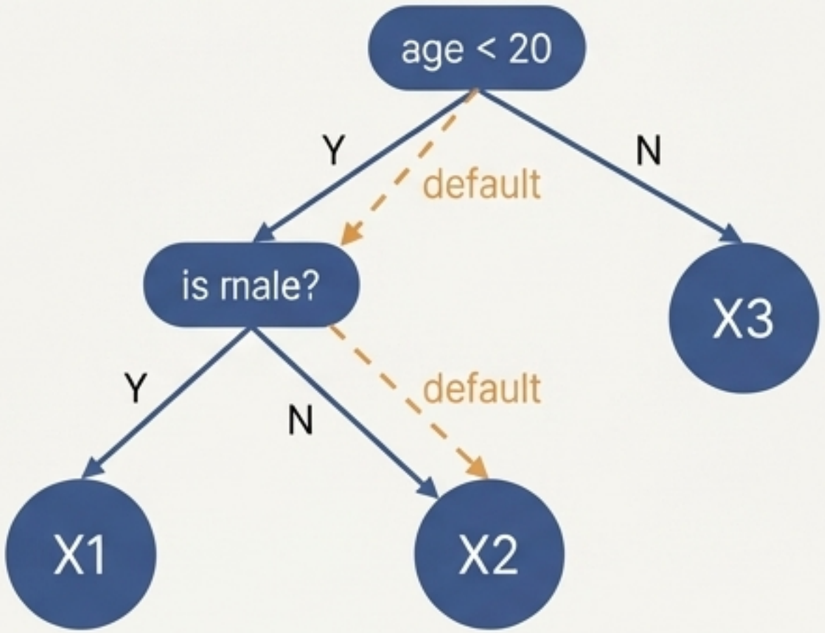
Algorithmic Innovation 1: Sparsity-awareness handles missing data natively.

The Problem

Real-world data is often sparse due to missing values, zero entries, or one-hot encoding. Naive implementations fail to handle this efficiently.

The Solution

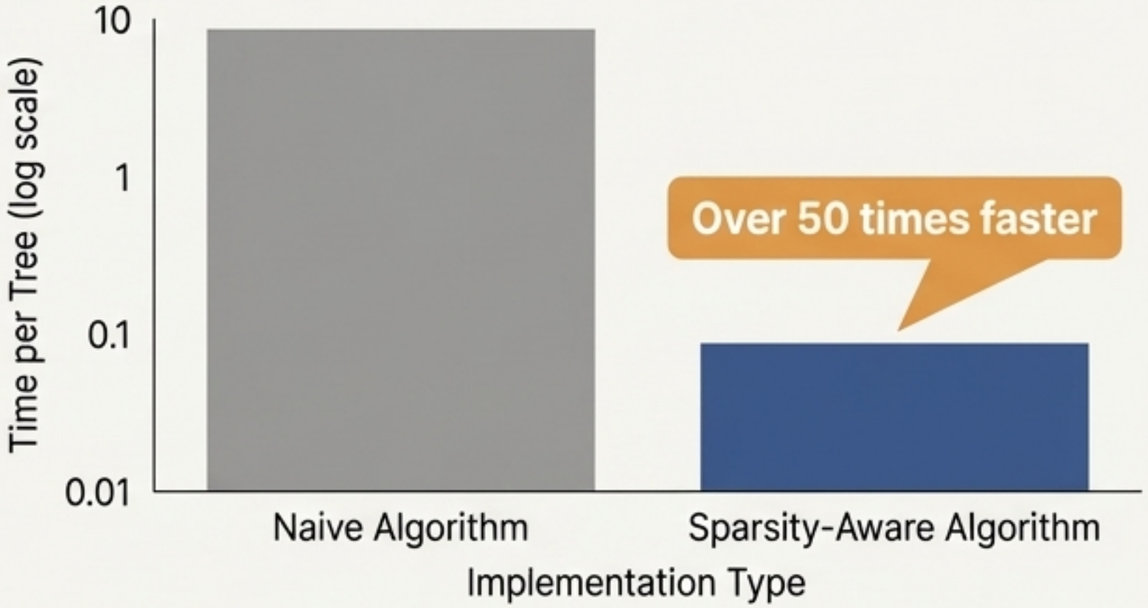
XGBoost introduces a “default direction” for each branch. When a feature value is missing, the instance is classified into this learned, optimal direction.



The Impact

The algorithm only visits non-missing entries, making its complexity linear to the number of non-missing entries. This provides a dramatic speedup.

Performance on Allstate-10K Dataset



Key Takeaway:
XGBoost natively handles missing data and sparse features, making it dramatically faster and more effective on real-world datasets.

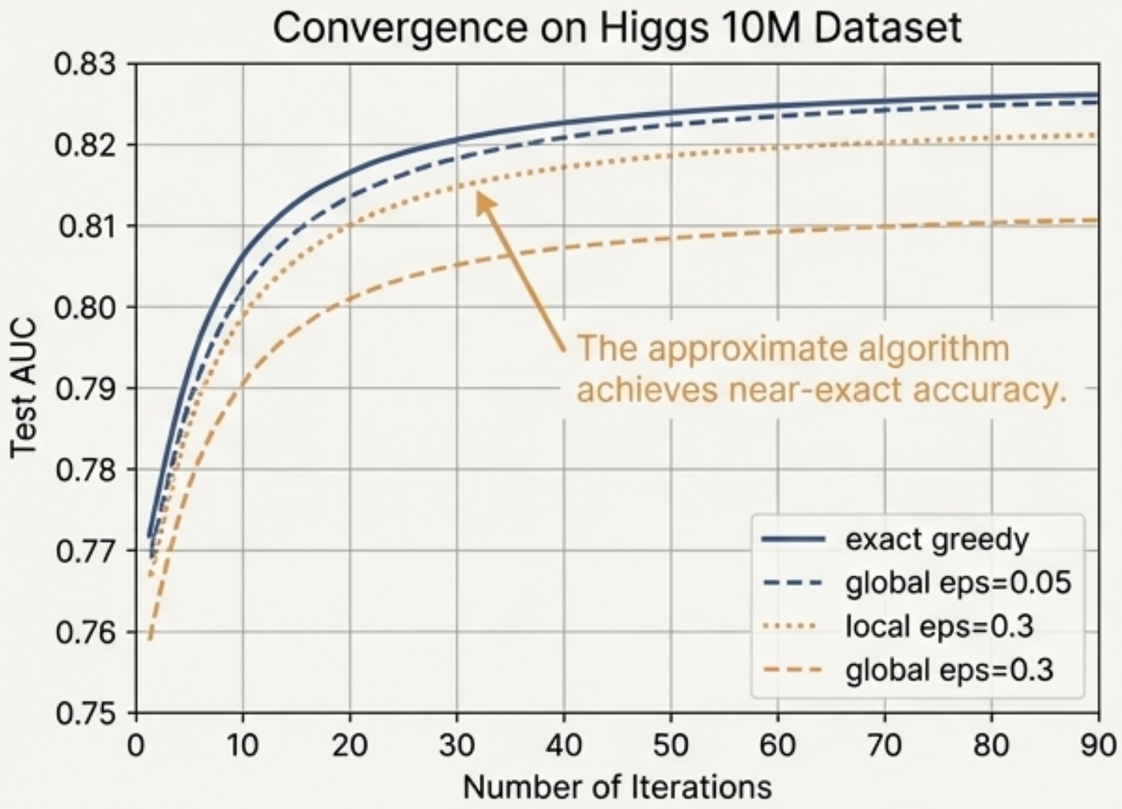
NotebookLM

Algorithmic Innovation 2: An approximate algorithm scales to massive datasets.

The Concept

- **The Challenge:** The exact greedy algorithm, which tests every possible split, is computationally prohibitive when data does not fit in memory.
- **The Solution:** XGBoost uses an approximate algorithm that proposes a limited number of candidate split points based on percentiles of the feature distribution. It then finds the best split among these proposals.
- **The Detail:** This is made robust by a novel **Weighted Quantile Sketch** algorithm, a theoretically-justified method for proposing candidates on weighted data (where weights h are derived from the second-order gradients).

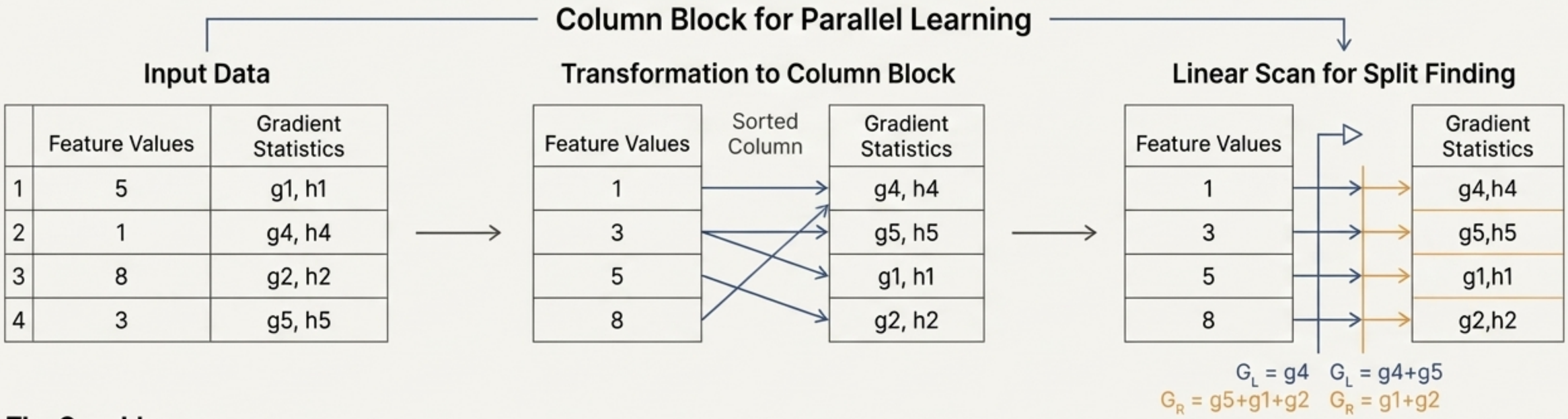
The Evidence



Key Takeaway: For datasets that don't fit in memory, XGBoost cleverly approximates splits, achieving near-exact accuracy at a fraction of the computational cost.

NotebookLM

The system's foundation is a data structure built for parallel learning.



The Core Idea

The most time-consuming part of tree learning is repeatedly sorting data. XGBoost solves this by pre-processing the data into an in-memory structure called a **Block**.

How it Works

1. Data is stored in compressed column (CSC) format.
2. Each column is sorted by its corresponding feature value.
3. Pointers are stored from feature values back to the instance indices.

Key Takeaway: Data is organised into in-memory blocks and pre-processed once. This saves enormous time on subsequent iterations by avoiding repeated sorting.

NotebookLM

System Innovation 3: Cache-aware prefetching minimises CPU bottlenecks.

The Problem

In the block structure, gradient statistics are fetched via indirect pointers, leading to non-contiguous memory access. When the data is large, this causes frequent CPU cache misses, which stall the processor.

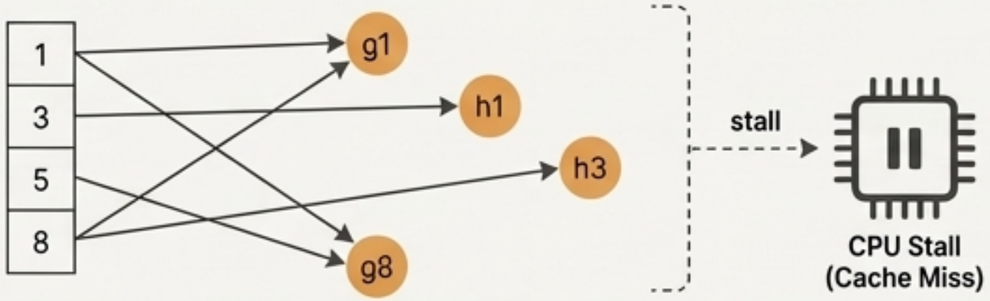
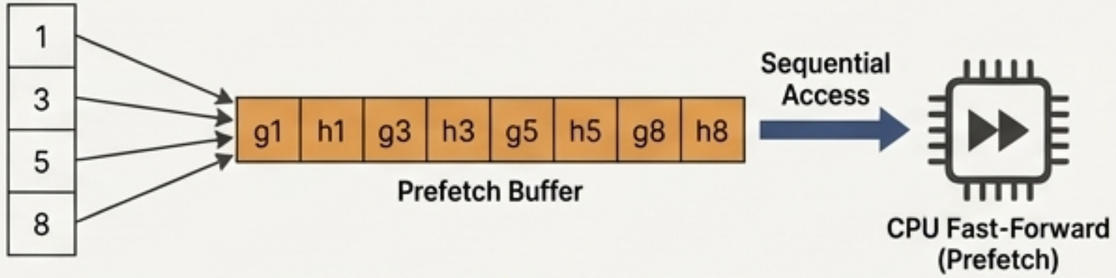


Figure 8 (simplified): Non-contiguous access to gradient statistics.

The Solution

XGBoost uses a cache-aware prefetching algorithm. Each thread allocates an internal buffer, fetches the required gradient statistics into it in a mini-batch, and then performs the accumulation.

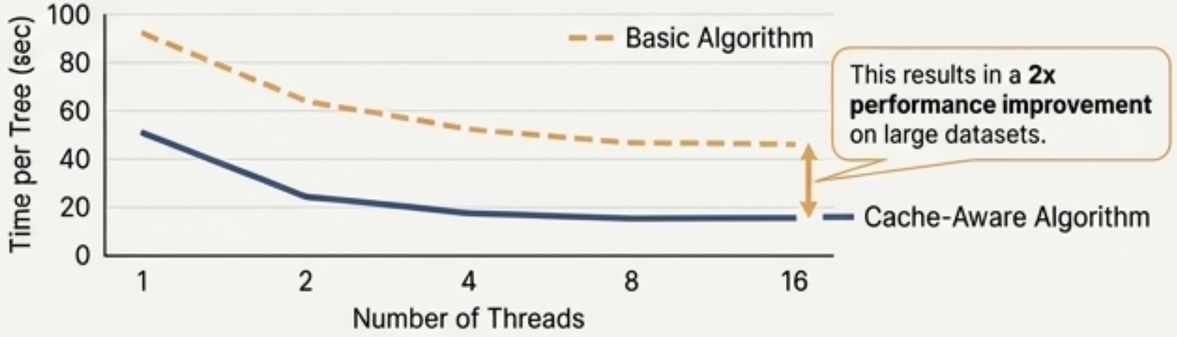


Contiguous prefetch buffer allows sequential access.

The Impact

Based on Figure 7. Inter charts a performance on Higgs 10M performant on dataset.

Figure 7: Performance on Higgs 10M Dataset



Key Takeaway: XGBoost minimises CPU cache misses—a common bottleneck—with a prefetching system that significantly speeds up computation on large datasets.

NotebookLM

System Innovation 4: Out-of-core computation processes billions of examples on a desktop.

The Challenge

How to train on datasets that are larger than the available RAM?

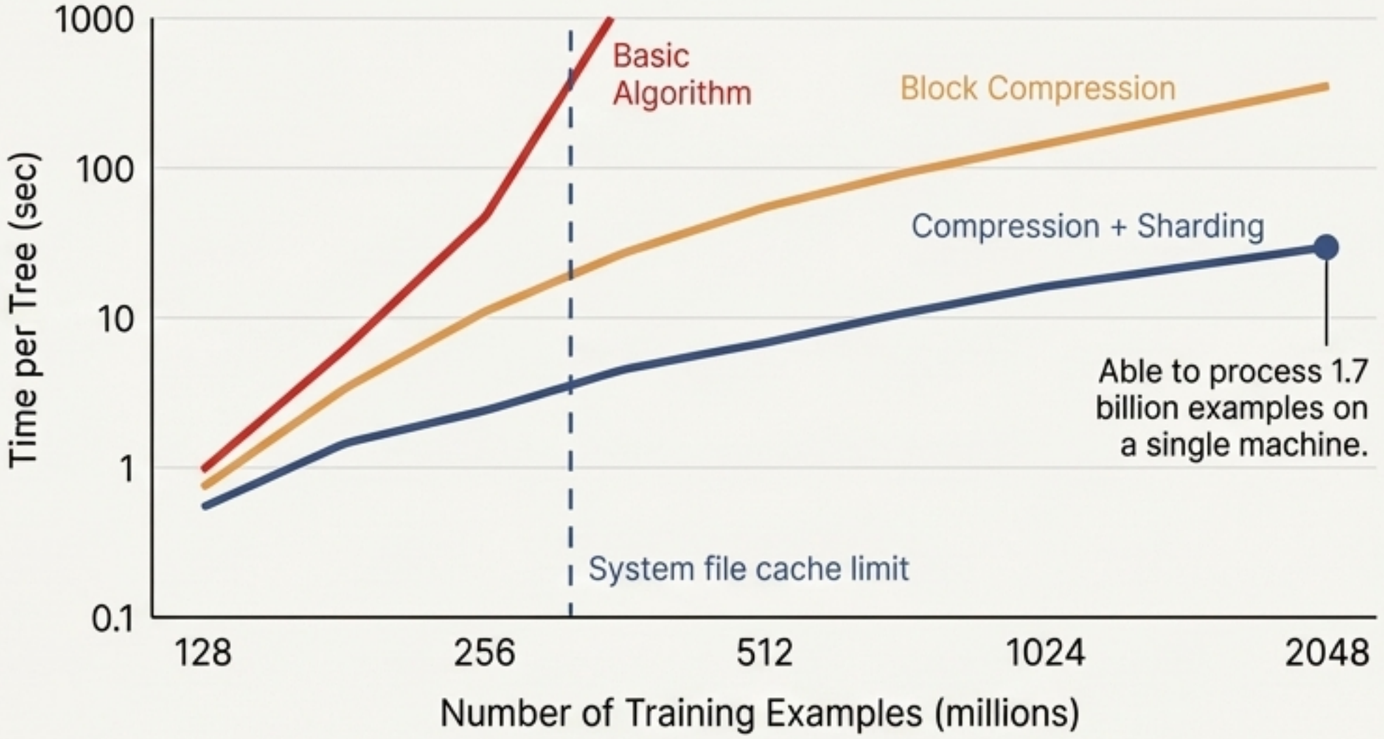
Block Compression

Data in each block is compressed by column and decompressed on-the-fly by an independent thread. This trades CPU cycles for reduced disk I/O. (Typical compression ratio: 26-29%).

Block Sharding

Data is sharded across multiple disks. A pre-fetcher thread for each disk reads data in parallel, increasing total disk throughput.

Out-of-Core Performance on Criteo Dataset



Key Takeaway: XGBoost can process billions of examples on a single desktop by intelligently compressing data and sharding it across multiple disks.

NotebookLM

The Verdict 1: It dominates standard libraries on a single machine.

Benchmark: Higgs Boson classification dataset (1 million instances), training 500 trees.

Conclusion:
XGBoost runs more than

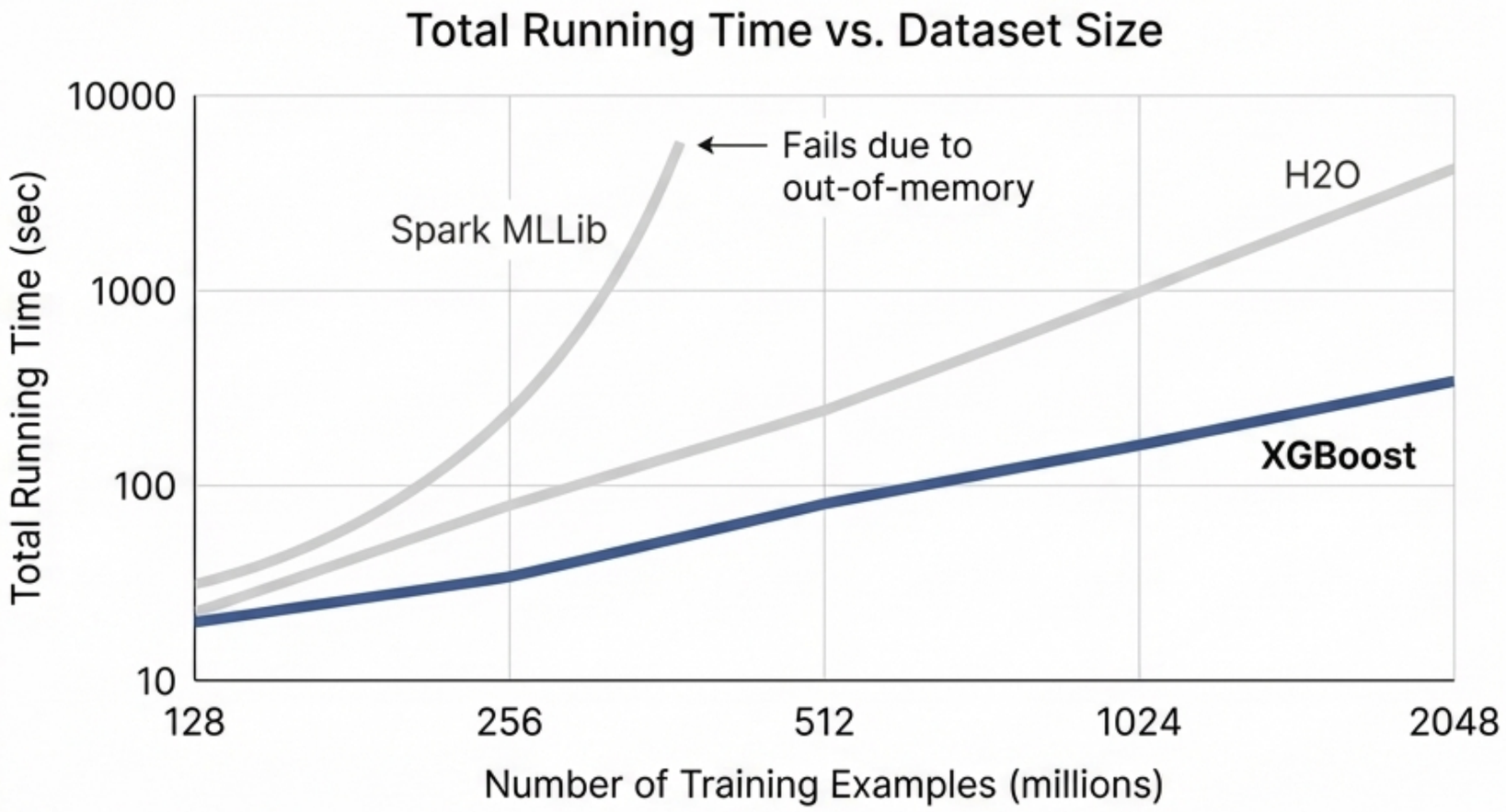
10x

faster than scikit-learn while delivering the same state-of-the-art accuracy (0.8304 vs 0.8302 AUC).

NotebookLM

The Verdict 2: It outperforms and out-scales production distributed systems.

Benchmark: Criteo terabyte click-through rate prediction on a 32-node cluster



Inter Medium

Conclusion: XGBoost is not only faster per iteration but is fundamentally more scalable, capable of solving problems that are intractable for other systems with the same hardware.

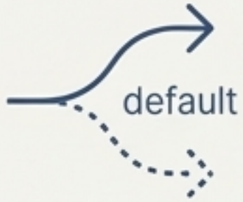
NotebookLM

The winning formula is a virtuous cycle of superior algorithms and purpose-built systems.

XGBoost's success is not a single feature, but the result of holistic design.

Superior Algorithms

- **Regularised Objective:** Controls complexity to prevent overfitting.

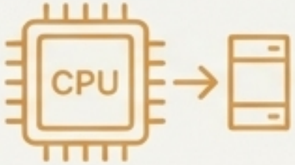


- **Sparsity-Aware Splits:** Handles missing data efficiently and dramatically accelerates training.

- **Weighted Quantile Sketch:** Enables accurate, scalable approximate learning.

Purpose-Built Systems

- **Column Blocks:** Minimises computation by pre-processing data for parallel access.



- **Cache-Aware Prefetching:** Optimises for hardware, doubling speed on large datasets.

- **Out-of-Core Computation:** Scales to billions of examples on minimal hardware.

NotebookLM