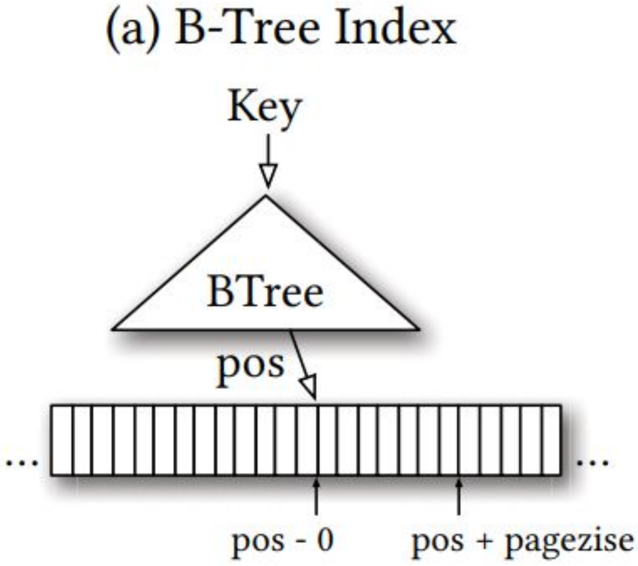
Meta-Learning-Based Adaptive Model Selection for Learned Indexes

Amay Dixit
12340220
DSL501 - Major Project

Motivation and Background

- **What are Database Indexes?**
  - Data structures that improve query performance
  - Act as "shortcuts" to find data quickly
  - Essential for modern database systems
- **Classic Approaches:**
  - B-Trees: Balanced tree structures
  - Hash Tables: Direct key-to-location mapping
  - Binary Search Trees: Hierarchical organization
- **The Challenge:**
  - Generic structures for all data types
  - Don't leverage data patterns
  - Performance plateaus with growing datasets



The Data Pattern Problem

Real-world data is heterogeneous:

- Geographic coordinates (clustered patterns)
- Timestamps (temporal trends)
- User IDs (sequential with gaps)
- Stock prices (volatile patterns)

Traditional indexes treat all data the same way

Opportunity: What if we could learn from data patterns?

Enter Learned Indexes - The Paradigm Shift

Revolutionary Idea (Kraska et al., 2018):

- Replace index structures with machine learning models
- Models learn data distribution patterns
- Predict approximate position of keys

How it works:

- Model learns mapping: Key → Approximate Position
- Use prediction to narrow search space
- Combine with small traditional structures for accuracy

Key Insight: Data has patterns, ML can exploit them

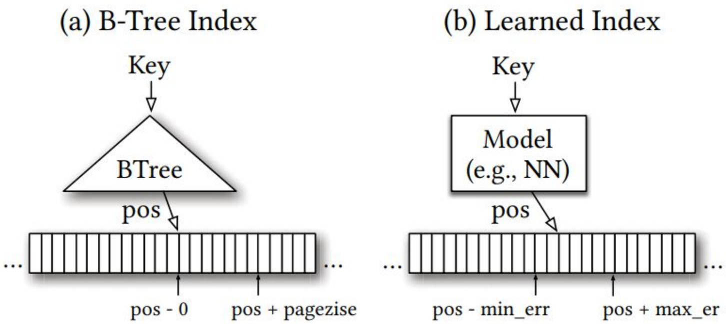


Figure 1: Why B-Trees are models

Learned Indexes vs Traditional - A Comparison

Traditional B-Tree:

- Fixed structure regardless of data
- O(log n) complexity always
- No adaptation to data patterns

Learned Index:

- Model adapts to data distribution
- Can achieve O(1) for well-structured data
- Leverages predictable patterns

Example: For sorted timestamps, a linear model might predict position perfectly, eliminating tree traversal entirely.

		Map Data			Web Data			Log-Normal Data		
Type	Config	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)	Size (MB)	Lookup (ns)	Model (ns)
Btree	page size: 32	52.45 (4.00x)	274 (0.97x)	198 (72.3%)	51.93 (4.00x)	276 (0.94x)	201 (72.7%)	49.83 (4.00x)	274 (0.96x)	198 (72.1%)
	page size: 64	26.23 (2.00x)	277 (0.96x)	172 (62.0%)	25.97 (2.00x)	274 (0.95x)	171 (62.4%)	24.92 (2.00x)	274 (0.96x)	169 (61.7%)
	page size: 128	13.11 (1.00x)	265 (1.00x)	134 (50.8%)	12.98 (1.00x)	260 (1.00x)	132 (50.8%)	12.46 (1.00x)	263 (1.00x)	131 (50.0%)
	page size: 256	6.56 (0.50x)	267 (0.99x)	114 (42.7%)	6.49 (0.50x)	266 (0.98x)	114 (42.9%)	6.23 (0.50x)	271 (0.97x)	117 (43.2%)
	page size: 512	3.28 (0.25x)	286 (0.93x)	101 (35.3%)	3.25 (0.25x)	291 (0.89x)	100 (34.3%)	3.11 (0.25x)	293 (0.90x)	101 (34.5%)
Learned Index	2nd stage models: 10k	0.15 (0.01x)	98 (2.70x)	31 (31.6%)	0.15 (0.01x)	222 (1.17x)	29 (13.1%)	0.15 (0.01x)	178 (1.47x)	26 (14.6%)
	2nd stage models: 50k	0.76 (0.06x)	85 (3.11x)	39 (45.9%)	0.76 (0.06x)	162 (1.60x)	36 (22.2%)	0.76 (0.06x)	162 (1.62x)	35 (21.6%)
	2nd stage models: 100k	1.53 (0.12x)	82 (3.21x)	41 (50.2%)	1.53 (0.12x)	144 (1.81x)	39 (26.9%)	1.53 (0.12x)	152 (1.73x)	36 (23.7%)
	2nd stage models: 200k	3.05 (0.23x)	86 (3.08x)	50 (58.1%)	3.05 (0.24x)	126 (2.07x)	41 (32.5%)	3.05 (0.24x)	146 (1.79x)	40 (27.6%)

Figure 4: Learned Index vs B-Tree

Current Learned Index Limitations

The One-Model Problem:

- Most current approaches use a single model type for entire dataset
- Examples: Always linear regression, always neural networks
- **Problem:** Real data has varying local patterns

Real-World Scenario: Uber Driver Location Database

📍 **Downtown Manhattan (High Density)**

- Pattern: tightly clustered coordinates (small increments)
- Current Model: Same as everywhere else (fails at fine-grained lookups)
- Optimal Model: High-precision Neural Network

🏘️ **Suburban Areas (Sparse)**

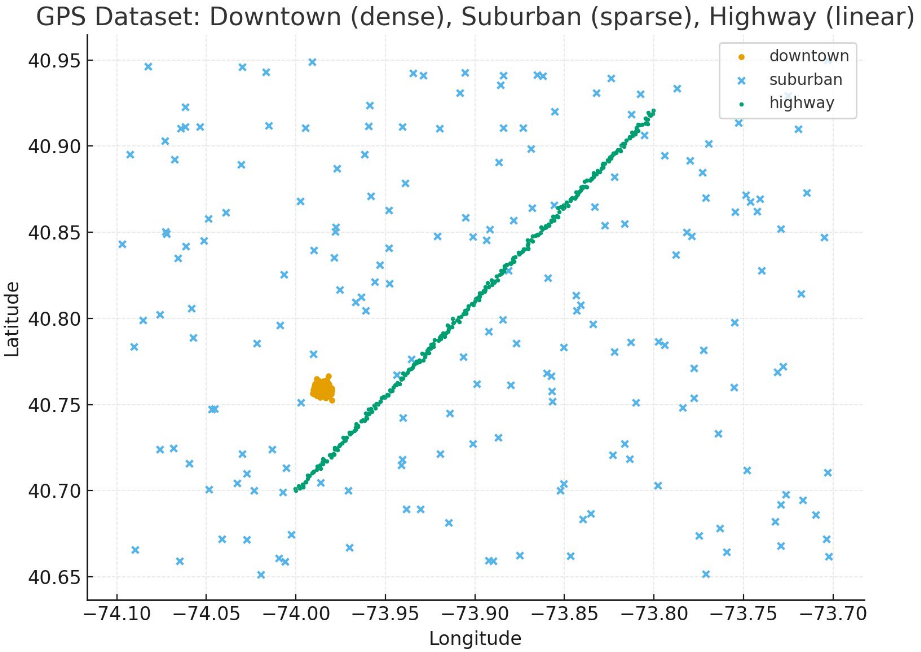
- Pattern: Large jumps between coordinates
- Current Model: Struggles with gaps (poor prediction)
- Optimal Model: Decision Tree (handles discontinuities)

🛣️ **Highway Corridors (Linear)**

- Pattern: Sequential progression along a route
- Current Model: Overcomplicated model (wasteful)
- Optimal Model: Linear Regression

Current approaches: Force one model type everywhere

Our insight: Different regions need different models



Problem Statement - Formal Definition

Research Question: Can we improve learned index performance by using meta-learning to adaptively select the optimal model type for each data segment?

Hypothesis: Different data patterns require different model architectures. A meta-learning approach that selects models based on segment characteristics will outperform uniform model architectures.

Success Metrics:

- 15-25% reduction in lookup times
- Improved robustness across diverse datasets
- Interpretable model selection patterns

Novelty - The Meta-Learning Solution

Meta-Learning Definition:

- **Core Concept:** Machine learning system that selects optimal algorithms for specific problem instances
- **Input:** Problem characteristics (features)
- **Output:** Algorithm/model recommendation
- **Training:** Learn from performance patterns across diverse problem types

Technical Process:

1. **Feature Extraction:** Characterize each problem instance
2. **Performance Mapping:** Record which algorithms work best for which features
3. **Meta-Model Training:** Learn the mapping from features to optimal algorithm
4. **Inference:** Predict best algorithm for new, unseen problems

Our Meta-Learning Application:

- **Problem Instance:** Each data segment in the index
- **Feature Space:** Statistical properties of data segments
- **Algorithm Space:** {Linear Regression, Polynomial, Decision Tree, Neural Network}
- **Meta-Learner:** Random Forest Classifier trained on segment features

Why Meta-Learning Here:

- **Heterogeneous Data:** Different segments have different optimal models
- **Automation:** Eliminates manual model selection and tuning
- **Scalability:** Decisions made automatically for thousands of segments
- **Adaptability:** System learns from new data patterns over time

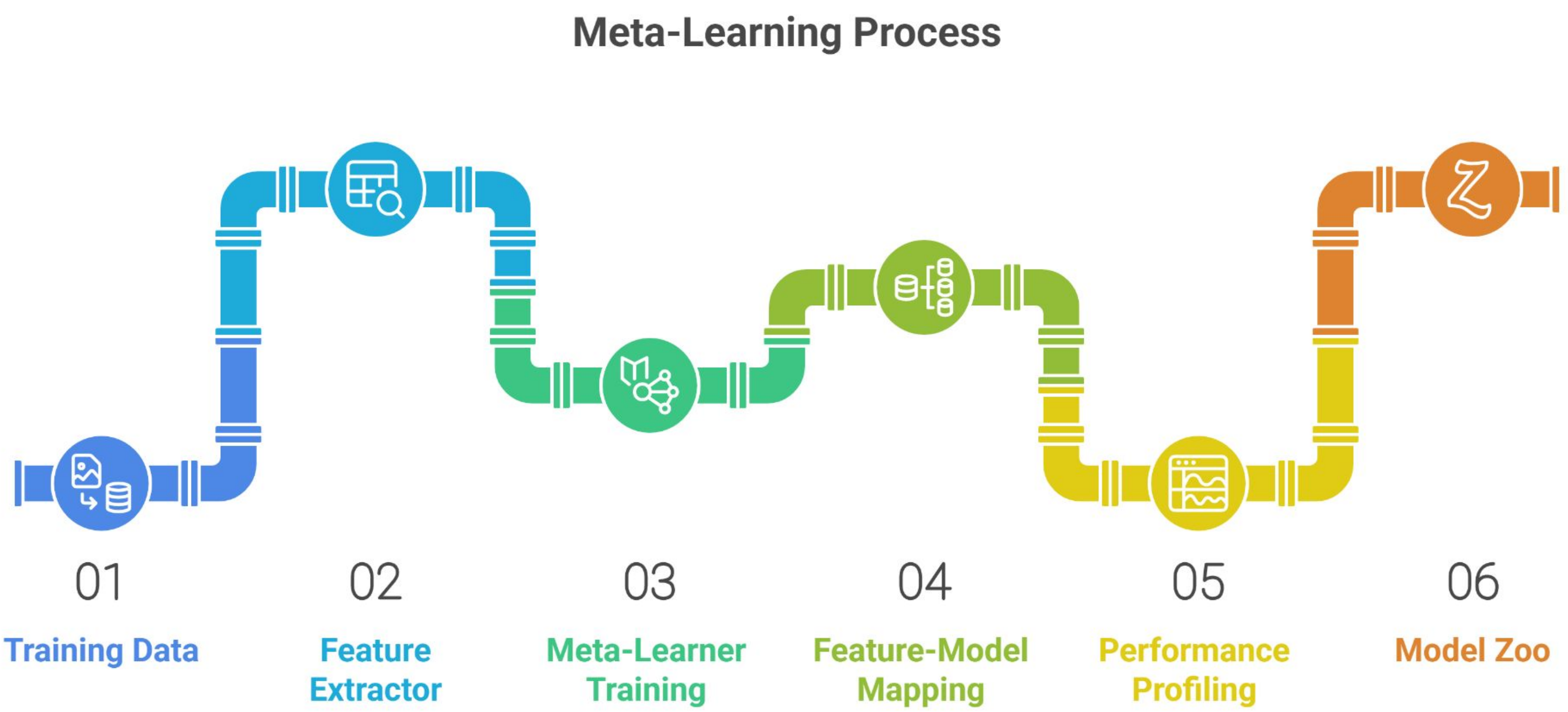
Key Innovation:

- **First application** of meta-learning to learned index structure optimization
- **Segment-level granularity** rather than dataset-level decisions
- **Feature-driven selection** based on statistical data characteristics

System Architecture Overview

Four Core Components:

1. **Feature Extractor**
   - Analyzes each data segment
   - Extracts statistical features (skewness, entropy, variance)
   - Captures local data characteristics
2. **Model Zoo**
   - Repository of candidate models
   - Linear, Polynomial, Decision Trees, Neural Networks
   - Pre-profiled for performance characteristics
3. **Meta-Learner**
   - Random Forest Classifier
   - Maps segment features → optimal model choice
   - Trained on diverse data patterns
4. **Adaptive Index**
   - Integrates all components
   - Per-segment model selection
   - Unified query interface



Meta-Learning Process
01
Training Data
02
Feature Extractor
03
Meta-Learner Training
04
Feature-Model Mapping
05
Performance Profiling
06
Model Zoo

Meta-Learning Process - Training the Decision Maker

Training Phase:

1. Generate diverse synthetic datasets
2. Segment each dataset
3. Extract features for each segment
4. Test all candidate models on each segment
5. Record optimal model for each feature pattern
6. Train meta-learner: Features → Best Model

Inference Phase:

1. New data segment arrives
2. Extract statistical features
3. Meta-learner predicts best model
4. Use predicted model for this segment

Key Advantage: Automated, data-driven decision making

Meta Learning is not Ensemble Learning

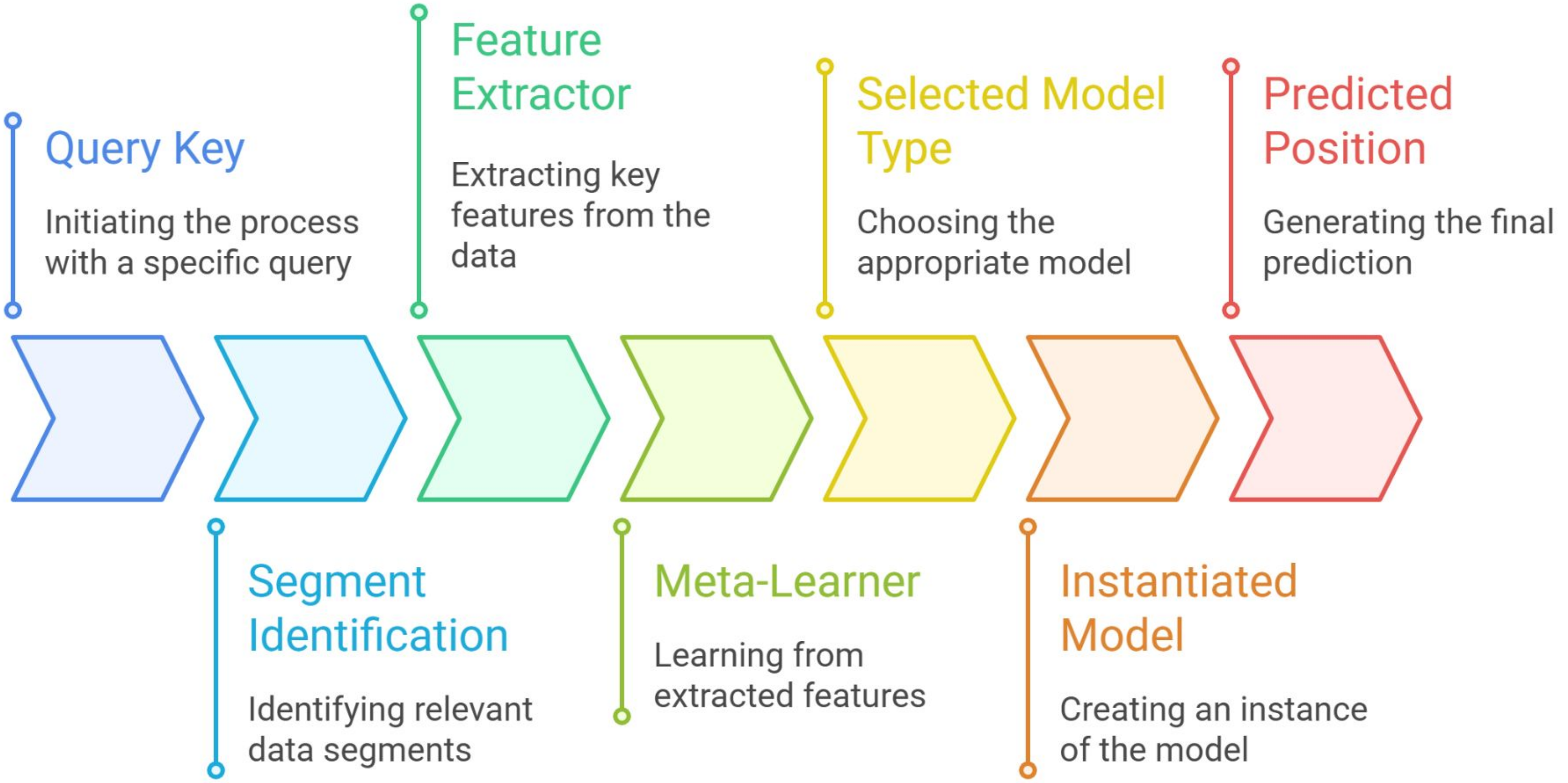
What Ensemble Learning Does:

- **Combines multiple models** to make a single prediction
- **All models run simultaneously** on the same data
- **Aggregates results** (voting, averaging, weighted combination)
- **Example:** Random Forest uses multiple decision trees and averages their predictions

What Meta-Learning Does:

- **Selects one optimal model** based on problem characteristics
- **Only the chosen model runs** on the data
- **Makes algorithmic choice** before prediction
- **Example:** Our system chooses between linear regression OR decision tree, not both

Predictive Modeling Workflow



Feature Engineering - Understanding Data Segments

Statistical Features Extracted:

- **Distribution Shape:** Skewness, Kurtosis
- **Variability:** Variance, Standard deviation
- **Information Content:** Entropy
- **Structure:** Gap density, monotonicity
- **Trend:** Local slope, curvature

Why These Matter:

- Skewed data → Tree models might excel
- Linear trends → Linear regression optimal
- High entropy → Neural networks needed
- Regular gaps → Polynomial approximation

Model Zoo - Diverse Approaches for Diverse Data

Candidate Models:

1. **Linear Regression**
   - Best for: Monotonic, linear trends
   - Complexity: O(1) prediction
   - Use case: Sequential IDs
2. **Polynomial Regression**
   - Best for: Smooth curves, growth patterns
   - Complexity: O(1) prediction
   - Use case: Exponential data growth

3. Decision Trees

- Best for: Irregular patterns, plateaus
- Complexity: O(log depth)
- Use case: Categorical-influenced data

4. Shallow Neural Networks

- Best for: Complex, non-linear patterns
- Complexity: O(hidden units)
- Use case: Highly irregular distributions

Methodology - Experimental Design & Baseline Comparisons

Dataset Strategy:

- **Synthetic Data:** Controlled pattern evaluation
  - Piecewise linear, polynomial, noisy patterns
  - Known ground truth for validation
- **Real-world Data:** Practical validation
  - OpenStreetMap coordinates (geographic patterns)
  - NYC Taxi timestamps (temporal patterns)
  - Stack Overflow IDs (user behavior patterns)

Evaluation Metrics:

- Lookup latency (primary)
- Memory consumption
- Model selection accuracy
- Robustness across data types

We'll compare against:

1. **Traditional B-Trees**
   - Industry standard
   - Consistent performance baseline
2. **Static Learned Indexes**
   - Single model type (e.g., all linear)
   - Current state-of-the-art approaches
3. **Oracle Selection**
   - Perfect model choice (theoretical optimum)
   - Upper bound for our approach

Success Criteria:

- Outperform B-Trees by >20%
- Outperform static learned indexes by >10%
- Achieve >80% of oracle performance

Current State-of-the-Art Limitations

- **CARMI: Heuristic rules → Our Work: ML-driven decisions**
- **RMI (Recursive Model Index):** Uses neural networks uniformly
- **ALEX:** Adaptive but limited model types
- **PGM-Index:** Only piecewise linear models
- **Gap:** No intelligent, automated model selection based on data characteristics

Approach	Model Types	Selection	Adaptation	Meta-Learning
Traditional	Fixed	None	None	No
RMI	Single	Manual	Limited	No
ALEX	Few	Heuristic	Some	No
PGM	One	None	Minimal	No
OURS	Multiple	ML	Full	YES

Segmentation Inspiration - CARMI's Success Story

CARMI Paper Key Findings:

- **Cache-Aware Recursive Model Index (2022)**
- **Demonstrated:** Different data regions benefit from different model types
- **Results:** 20-40% performance improvement through adaptive model selection
- **Limitation:** Used heuristic-based selection, not machine learning

What CARMI Showed Us:

- Traditional Approach: [Single Model] applied to [Entire Dataset]
- CARMI Approach: [Different Models] for [Different Regions]
- Our Meta-Learning: [ML-Selected Models] for [Statistically-Defined Segments]

Why Segmentation Works:

- **Real data is heterogeneous** - different patterns in different regions
- **Local optimization** outperforms global optimization
- **Segment-level decisions** allow fine-grained performance tuning

workloads	indexes	uniform dataset			lognormal dataset		
		space /MB	time /ns		space /MB	time /ns	
			uniform	zipfian		uniform	zipfian
read-only	CARMI	**1394.3**	**114.3**	**85.1**	**1475.7**	**129.2**	**92.1**
	ALEX	1474.3	112.5	83.3	1476.7	115.5	89.3
	B-Tree	2276.0	482.6	267.8	2276.0	480.6	268.1
write-heavy	CARMI	**1509.9**	**168.4**	**154.5**	**2127.4**	**190.3**	**168.6**
	ALEX	1474.3	164.1	147.0	1476.7	294.0	262.9
	B-Tree	2276.0	500.3	419.9	2276.0	522.4	421.9
read-heavy	CARMI	**1509.4**	**135.8**	**102.1**	**2077.5**	**155.0**	**104.2**
	ALEX	1474.3	116.2	89.2	1476.7	121.7	95.4
	B-Tree	2276.0	488.9	286.8	2276.0	483.4	292.5
range scan	CARMI	**1509.4**	**402.4**	**256.4**	**2077.5**	**434.3**	**271.3**
	ALEX	1474.3	375.0	233.6	1476.7	383.6	240.3
	B-Tree	2276.0	704.5	416.5	2276.0	728.5	414.7

YCSB dataset		OSMC dataset			Face dataset		
space /MB	time /ns	space /MB	time /ns		space /MB	time /ns	
			uniform	zipfian		uniform	zipfian
11.4*	**93.2**	**1771.5**	**287.5**	**160.1**	**1812.6**	**248.5**	**166.4**
1474.3	97.6	1522.0	421.9	221.4	1522.0	404.0	228.1
2276.0	269.7	2276.0	479.9	262.5	2276.0	480.6	261.5
11.4*	**161.2**	**1772.5**	**357.9**	**246.0**	**1882.7**	**316.9**	**246.2**
1474.3	317.3	1522.0	1109.1	893.3	1522.0	1801.7	1777.6
2276.0	401.2	2276.0	503.5	421.3	2276.0	498.7	432.9
11.4*	**104.4**	**1771.9**	**330.2**	**172.5**	**1812.9**	**278.3**	**174.2**
1474.3	234.7	1522.0	651.7	420.4	1522.0	993.8	829.0
2276.0	280.4	2276.0	483.1	284.5	2276.0	477.6	280.1
11.4*	**266.4**	**1771.9**	**585.6**	**305.3**	**1812.9**	**600.9**	**323.9**
1474.3	359.3	1522.0	935.3	598.2	1522.0	1246.4	990.5
2276.0	393.1	2276.0	696.4	417.1	2276.0	698.7	415.3

Novelty

- First meta-learning application to learned indexes
- Segment-level model selection (vs. dataset-level)
- Feature-driven automation (vs. heuristic-based)
- Statistical characterization of optimal model patterns

Challenges

Challenge 1: Feature Engineering Complexity

→ Solution: Start with proven statistical features, expand iteratively

Challenge 2: Segmentation Strategy

→ Solution: Test multiple approaches (fixed-size, pattern-based, adaptive)

Challenge 3: Meta-Model Generalization

→ Solution: Comprehensive synthetic training data + cross-validation

Challenge 4: Performance Overhead

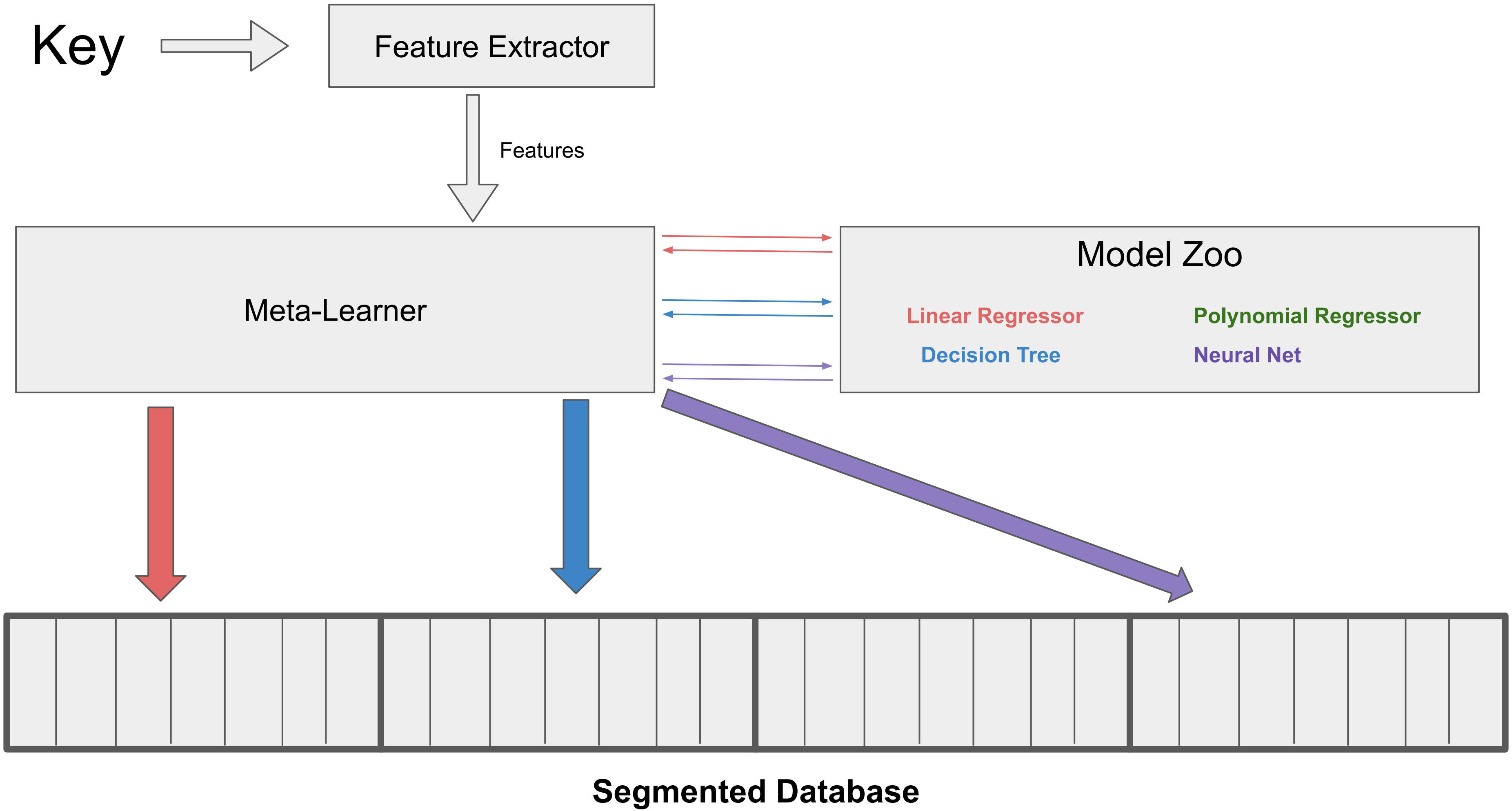
→ Solution: Efficient caching + batch processing of segments

Limitations

- Inserts and Updates not optimized
- Offline learning -> could be expanded to online learning in the future
- Haven’t thought of cache based optimizations
- Training Cost for Meta-Learner?
- Limited Fault Tolerance -> can have fallback mechanisms later, may introduce overhead
- Complexity of Deployment
  - A database admin can easily configure a B-Tree or Hash index.
  - Deploying a meta-learning-based index requires ML infrastructure, retraining pipelines, and monitoring — which increases system complexity.

Thank You

Open for Questions & Discussion


Key
Feature Extractor
Features
Meta-Learner
Model Zoo
Linear Regressor
Polynomial Regressor
Decision Tree
Neural Net
Segmented Database