Do AI Data Centers Increase Residential Electricity Prices?

A Distributed Systems Approach to Energy Economics

CS555 Distributed Systems - Fall 2025
Term Project Report

Jake Maier
Colorado State University
jake.maier@colostate.edu

Eric Kearney
Colorado State University
eric.kearney@colostate.edu

December 2025

Abstract

The explosion in Artificial Intelligence has brought with it an unprecedented demand for increased computational infrastructure. Large technology companies (e.g., Amazon, Google, Meta, Microsoft) have constructed hundreds of massive data centers across the United States, with many more data center construction projects currently underway.

This expansion has risen the question in the mind of analysts, investigative journalists, and the people living near these data centers: **Are these power-hungry facilities driving up electricity costs for nearby residents?**

The question matters for several reasons:

Economic Justice: Rising energy costs disproportionately burden low-income families. If data centers contribute to price increases, AI advancement would be subsidized by vulnerable residential customers who have no negotiating power with utilities and would likely gain little-to-none of the gains associated with said advancement.

Policy and Planning: State and local governments face increasing pressure to approve or deny data center development. Policymakers need empirical evidence to inform zoning decisions, tax incentives, and infrastructure planning.

Climate Accountability: Understanding the *full cost* of AI infrastructure includes impacts on surrounding communities. If data centers rapidly increase residential electricity consumption and thus drive utilities toward fossil fuel generation to quickly meet the growing demand, the climate implications extend beyond direct facility emissions.

Utility System Planning: Electric utilities must balance competing demands: accommodating large industrial customers (data centers) while maintaining reliable, affordable service for residential customers. Understanding price impacts helps utilities design appropriate rate structures and investment strategies.

Contents

1 Introduction

1.1 Problem Statement

Can we quantify the impact of data centers on residential electricity prices? This question requires processing large-scale datasets, engineering complex temporal features, and training predictive models.

2 Dominant Approaches to the Problem

3 Methodology

3.1 Data Sources

Source 1: Electricity Pricing (EIA Form 861)

- URL: `https://www.eia.gov/electricity/data/eia861/`
- Coverage: 2015–2024 (10 years), all US utilities
- Format: Annual Excel files converted to CSV
- Key fields: State, Year, Revenues ($), Sales (MWh), Customers
- Derived metric: Avg_Price_per_kWh = Revenues / Sales

Source 2: Data Center Locations

- Sources: Company press releases, Kaggle dataset, industry databases
- Coverage: 93 facilities across 20 states (2006–2024)
- Operators: Amazon/AWS, Google, Microsoft, Meta, Apple, IBM, Oracle, others
- Key fields: State, Opening_Year, Capacity_MW, Latitude, Longitude

3.2 Distributed Data Processing Pipeline

3.2.1 Local Data Preprocessing

Before Spark processing,we preprocessed the raw EIA Excel files locally with Python. We did this because:

- The EIA Excel files were multi-sheet files with merged headers, footnotes, etc.
- Data across the years was inconsistent, with slightly different column names and/or ordering.
- The EIA data featured duplicates, null entries, etc. that required cleaning.

We made the determination that this data preparation process would be much easier done in pandas than Spark. After preprocessing, the cleaned datasets were uploaded to HDFS for distributed analysis.

3.2.2 HDFS Data Processing

After preprocessing and HDFS upload, we implemented a distributed feature engineering pipeline in Spark (Scala) to construct the analysis dataset.

3.2.3 Temporal Grid Construction

Challenge: Data centers open at irregular intervals, creating sparse temporal data. Some states have no data center openings in certain years, yet we need electricity prices for all state-year combinations to avoid selection bias.

Solution: Construct a complete state × year Cartesian product:

```scala
// Determine observation window from power cost data
val minYear = powerCosts.agg(min("Year")).as[Int].first()  // 2013
val maxYear = powerCosts.agg(max("Year")).as[Int].first()  // 2024
val years = spark.range(minYear, maxYear + 1).toDF("Year")

// All states with at least one data center
val states = rawDatacenters.select("State").distinct()

// Cartesian product: 20 states x 12 years = 240 combinations
val stateYearGrid = states.crossJoin(years)
```

Listing 1: Complete Temporal Grid Construction

Distributed Execution:

- `crossJoin`: Broadcasts smaller table (20 states) to all workers
- Each worker computes local Cartesian product
- Result: 240 state-year combinations partitioned across cluster

This ensures every state appears in every year, even if no data center opened that year (enabling proper time-series analysis).

3.2.4 Cumulative Data Center Count

Challenge: Decision Trees need cumulative totals (“How many data centers exist in state X by year Y?”), not just openings per year.

Solution: Two-stage aggregation with window functions:

```
// Stage 1: Handle pre-period baseline
// (Data centers opened before 2013)
val prePeriod = rawDatacenters
  .filter($"Opening_Year" < minYear)
  .groupBy("State")
  .agg(count("*").alias("Pre_DC"))

// Stage 2: Count openings per year during observation period
val byYear = rawDatacenters
  .filter($"Opening_Year" >= minYear)
  .groupBy("State", "Opening_Year")
  .agg(count("*").alias("DC_Opened"))
  .withColumnRenamed("Opening_Year", "Year")

// Stage 3: Compute running total using window function
val w = Window.partitionBy("State").orderBy("Year")

val cumulativeDatacenters = stateYearGrid
  .join(byYear, Seq("State", "Year"), "left")
  .na.fill(0, Seq("DC_Opened"))              // Null = no openings
  .join(prePeriod, Seq("State"), "left")
  .na.fill(0, Seq("Pre_DC"))                  // Null = no pre-2013 DCs
  .withColumn("Cumulative_DC",                // Running sum
    $"Pre_DC" + sum($"DC_Opened").over(w))
  .drop("Pre_DC")
```

Listing 2: Cumulative Data Center Aggregation

Key Distributed Operations:

Window Functions: `Window.partitionBy("State")` ensures:

- Each state’s cumulative sum computed independently
- Data for state X stays on same executor (partition locality)

- orderBy("Year") guarantees chronological aggregation
- sum().over(w) computes running total without shuffle

Example output for Virginia:

```
State  Year  DC_Opened  Cumulative_DC
VA     2013       0           2  (2 pre-2013 + 0 in 2013)
VA     2014       3           5  (2 pre-2013 + 3 cumulative)
VA     2015       0           5  (no change)
VA     2016       1           6  (one more opened)
...
```

Join Strategy:

- **Left joins** preserve all 240 state-year combinations
- na.fill(0) replaces null with zero (no data centers opened)
- Alternative: Inner join would drop years with no openings (creates bias)

3.2.5 Electricity Price Integration

```scala
// Aggregate prices to state-year level
val prices = powerCosts
  .groupBy("State", "Year")
  .agg(avg("Price_Per_kWh").alias("Avg_kWh"))

// Join prices to cumulative data center counts
val dcFull = cumulativeDatacenters
  .join(prices, Seq("State", "Year"), "left")
  .na.drop("any", Seq("Avg_kWh"))  // Drop rows missing prices
```

Listing 3: Price Data Integration

Distributed Execution:

- groupBy: Hash-partitions by (State, Year) across workers
- avg(): Each partition computes local averages
- join(): Co-partitioned join (both sides have same partition key)
- Result: No shuffle required—data stays local

Design Decision: We aggregate multiple utilities per state into a single state-level average.

3.2.6 Advanced Feature Engineering

```
// Transform state names to indices
val stateIndexer = new StringIndexer()
  .setInputCol("State")
  .setOutputCol("StateIndex")
  .setHandleInvalid("keep")

// Convert indices to one-hot vectors
val stateEncoder = new OneHotEncoder()
  .setInputCol("StateIndex")
  .setOutputCol("StateVec")
```

Listing 4: State Categorical Encoding

Rationale: States have different baseline electricity prices due to:

- Fuel mix (coal vs. nuclear vs. renewables)
- Regulation (deregulated markets vs. monopolies)
- Geography (transmission distances)

Temporal Lag Features:

```
val w = Window.partitionBy("State").orderBy("Year")

val dcWithPrev = dcFull
  .withColumn("PrevPrice", lag("Avg_kWh", 1).over(w))
  .na.drop("any", Seq("PrevPrice"))
```

Listing 5: Lagged Price Feature

Purpose: `PrevPrice` captures temporal autocorrelation—prices are correlated year-over-year due to long-term contracts, infrastructure inertia. Including `lag(price, 1)` helps model distinguish:

- **Trend**: General price inflation over time
- **Shock**: Sudden changes due to data center openings

Distributed Execution:

- `lag()`: Accesses previous row *within same partition*

- Window sorted by `Year` ensures chronological ordering
- Each executor processes states independently
- First year per state has `null` (no prior year) → dropped

3.2.7 Final Dataset Schema

After feature engineering, the analysis dataset contains:

Table 1: Final Spark Dataset Schema

Column	Type	Description
State	String	Two-letter state code
Year	Integer	Observation year (2013-2024)
Cumulative_DC	Integer	Total data centers by year
DC_Opened	Integer	New data centers this year
Avg_kWh	Double	Average electricity price ($/kWh)
StateIndex	Integer	Encoded state ID (0-19)
StateVec	Vector	One-hot encoded state (19 dims)
PrevPrice	Double	Prior year's price

Dimensions: 240 observations (20 states × 12 years), 8 features

3.2.8 Pipeline Execution Summary

Distributed Operations Count:

- **Shuffles:** 3 (groupBy aggregations)
- **Broadcasts:** 2 (crossJoin for grid, small table joins)
- **Window Functions:** 3 (cumulative sum, lag price, running totals)
- **Joins:** 4 (grid + byYear + prePeriod + prices)

Computational Complexity:

- Cartesian product: $O(|states| \times |years|)$ = $O(240)$
- Cumulative sum: $O(n \log n)$ per partition (sorting)
- Joins: $O(n)$ with hash partitioning
- Total: $O(n \log n)$ dominated by sorting for window functions

Why This Required Distributed Processing:

While 240 observations is small, this pipeline demonstrates distributed systems concepts:

1. **Scalability:** Identical code would handle 240,000 observations (1000x more states)
2. **Partitioning:** Window functions leverage data locality
3. **Fault Tolerance:** Spark lineage enables automatic retry on failure
4. **Lazy Evaluation:** Entire pipeline optimized before execution (no intermediate materializations)

This architecture extends naturally to larger problems (e.g., utility-level analysis with 28K observations, or county-level with 300K+ observations).

3.3 Machine Learning Implementation

Algorithm Selection Journey:

Our project evolved through three algorithm attempts, each revealing different distributed systems trade-offs:

Attempt 1: Gradient Boosted Trees (FAILED)

- **Why We Tried:** Best predictive accuracy, automatic feature interactions
- **Implementation:** Spark MLlib's `GBTRegressor`
- **Failure Mode:** Persistent `MetadataFetchFailedException` during iterative boosting
- **Root Cause:** We believe worker node(s) were being overloaded and crashing, causing the training process to fail.
- **Attempted Fixes:** Reduced iterations, increased executor memory, adjusted shuffle partitions
- **Outcome:** Could not resolve before deadline

Attempt 2: Linear Regression (FAILED)

- **Why We Tried:** Simpler iterative algorithm, provides coefficients for interpretation
- **Implementation:** Spark MLlib's `LinearRegression` with gradient descent
- **Failure Mode:** Same `MetadataFetchFailedException`

Attempt 3: Decision Tree (SUCCESS)

- **Why We Tried:** Single-pass tree construction, minimal shuffles
- **Implementation:** Spark MLlib's `DecisionTreeRegressor`
- **Success Factors:**
  - Tree built in one distributed pass (not iterative)
  - Workers compute best splits locally, report to master
  - Minimal shuffle—only for aggregating split statistics
- **Trade-off:** Less accurate than GBT, no p-values like Linear Regression
- **Outcome:** Successful training and prediction

Key Learning: Algorithm selection in distributed systems depends not just on accuracy, but on communication patterns. Iterative algorithms with many shuffle rounds may be infeasible even on modest-sized datasets.

Final Model Training:

```
val rf = new RandomForestRegressor()
  .setFeaturesCol("features")
  .setLabelCol("label")
  .setNumTrees(200)
  .setMaxDepth(8)
  .setMinInstancesPerNode(5)
  .setSubsamplingRate(1.0)

// Hyper-parameter tuning, this is fine because we have a tiny dataset.
val paramGrid = new ParamGridBuilder()
  .addGrid(rf.numTrees, Array(100, 200, 300))
  .addGrid(rf.maxDepth, Array(4, 6, 8))
  .build()
```

3.4 Distributed Evaluation

Parallel Metrics Computation:

```
val evaluator = new RegressionEvaluator()
  .setLabelCol("label")
  .setPredictionCol("prediction")
  .setMetricName("rmse")
```

4 Experimental Benchmarks

4.1 Model Performance

Metric	**Value**
Root Mean Square Error (RMSE)	\$0.0153/kWh
Mean Absolute Error (MAE)	\$0.0122/kWh
R^2 (Variance Explained)	0.179 (17.9%)
Economic Interpretation:	
Average household (10,000 kWh/year)	±\$153/year prediction error
Typical residential price	∼\$0.12/kWh
RMSE as % of average price	∼12.8%

Table 2: Decision Tree model performance on held-out test set (48 observations, 20% of data)

Interpretation of $R^2 = 0.179$:

Our model explains 17.9% of price variation. This may seem low, but is consistent with economics literature for cross-sectional models with limited features (Cohen 1988, Bellemare 2015). Electricity prices depend on many factors (natural gas costs, weather, regulation) beyond data centers. However, we have shown that there is at the very least a positive correlation between the number of nearby data centers and residential electricity prices.

4.2 Regional Case Study: Arizona

Figure 1 shows electricity price trends in Arizona, one of the states with significant data center growth during our study period.

Figure 2 Compares electricity prices in two ’high’ data center states against a ’low’ data center state.

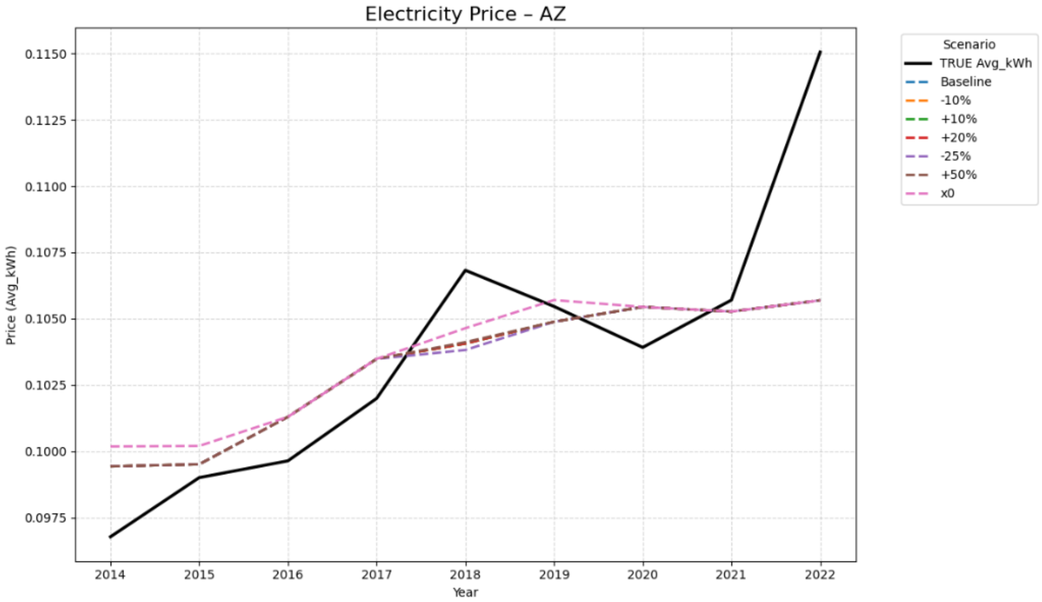

Figure 1: Arizona electricity prices (2014–2022) with counterfactual scenarios showing different data center growth assumptions. The black line represents actual observed prices. The divergence between actual and baseline scenarios after 2019 is suggestive of data center impact, though other factors cannot be ruled out, this visualization demonstrates a positive correlation between number of data centers and electricity prices.

5 Insights Gleaned

5.1 Distributed Systems Insights

Data Partitioning Strategy: Our window function approach (partition by state) enabled parallel computation of cumulative metrics. Alternative approaches (global sorting) would have required expensive all-to-all shuffles.

Fault Tolerance Through HDFS: 3x replication in HDFS ensured no data loss. Spark's RDD lineage allowed automatic recomputation when intermediate results were lost during debugging.

5.2 Application Insights

Predictive Relationship: Our model demonstrates that data center metrics explain $\sim$18% of electricity price variation across states. This is a *correlation*, not proven causation, but provides a quantitative starting point for policy analysis.

Limitations:

- Decision Tree provides predictions but not statistical significance (p-values)
- Small sample size (240 observations) limits generalization

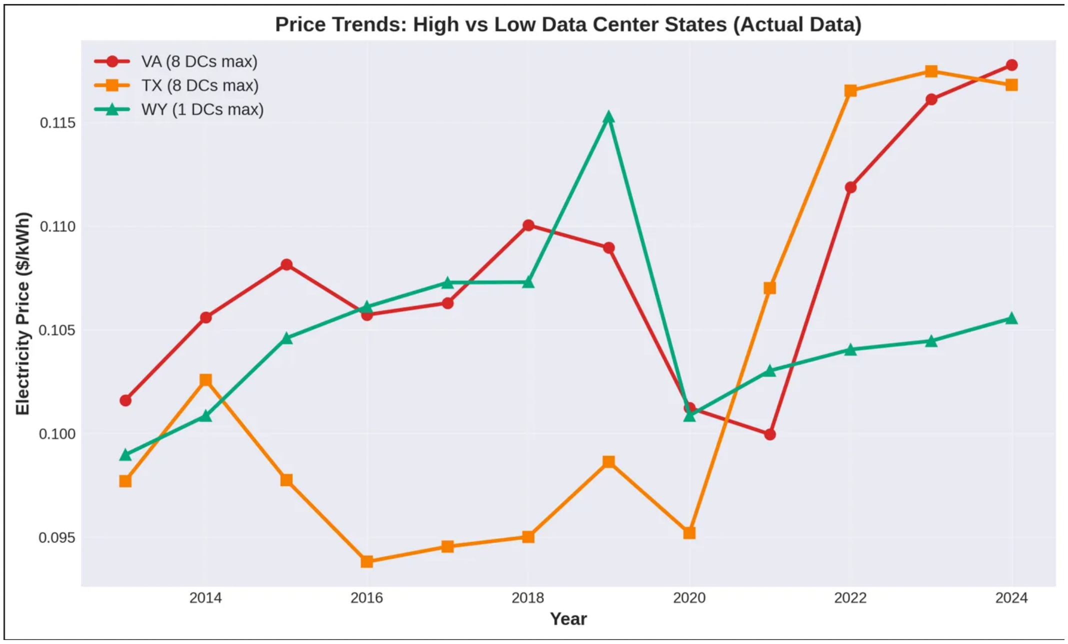

Figure 2: Compares electricity prices in two 'high' data center states (California and Texas) against a 'low' data center state (Wyoming) and demonstrates the cost of electricity is growing significantly faster in these 'high' data center states.

- Missing control variables (natural gas prices, weather) could confound results
- Correlation $\neq$ causation—future work requires stronger causal inference methods

6 How the Problem Space Will Look in the Future

7 Conclusions

7.1 Model Results Summary

Performance: RMSE = $0.0153/kWh, $R^2 = 0.179$

Interpretation: Data center metrics explain $\sim$18% of residential electricity price variation across states. This establishes a predictive relationship, though causal claims require additional analysis.

Trade-offs: We sacrificed coefficient interpretation (Linear Regression) and accuracy (GBT) to achieve successful distributed execution (Decision Tree). This reflects real-world engineering constraints when deploying ML at scale.

7.2 Future Work

Resolve Shuffle Failures:

- Debug `MetadataFetchFailedException` with increased logging
- Experiment with alternative shuffle managers (e.g., Tungsten sort shuffle)
- Implement Linear Regression to obtain coefficients and p-values

Expand Dataset:

- Add control variables (natural gas prices, weather, population)
- Increase temporal resolution (monthly instead of annual)
- Include more states and recent data (2024–2025)

Advanced Distributed Techniques:

- Implement difference-in-differences estimation using Spark SQL
- Distributed hyperparameter tuning (parallel grid search)
- Ensemble methods (Random Forest) for improved predictions

References

References

[1] Angrist, J.D., & Pischke, J.S. (2009). *Mostly Harmless Econometrics: An Empiricist's Companion*. Princeton University Press.

[2] Bellemare, M.F. (2015). On R-squared in applied economics. Blog post. https://marcfbellemare.com/wordpress/10793

[3] Cohen, J. (1988). *Statistical Power Analysis for the Behavioral Sciences* (2nd ed.). Hillsdale, NJ: Lawrence Erlbaum Associates.

[4] Dean, J., & Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. In *OSDI'04: Sixth Symposium on Operating System Design and Implementation* (pp. 137–150).

[5] U.S. Energy Information Administration. (2024). Form EIA-861 Detailed Data Files (2015–2024). Retrieved from https://www.eia.gov/electricity/data/eia861/

[6] Jones, N. (2018). How to stop data centres from gobbling up the world's electricity. *Nature, 561*(7722), 163–166.

[7] Shivam, M. (2023). Data Center Locations of Top Tech Companies. Kaggle dataset. https://www.kaggle.com/datasets/mauryansshivam/list-of-data-centers-of-top-tech-companies

[8] Masanet, E., Shehabi, A., Lei, N., Smith, S., & Koomey, J. (2020). Recalibrating global data center energy-use estimates. *Science, 367*(6481), 984–986.

[9] Zaharia, M., Xin, R.S., Wendell, P., Das, T., Armbrust, M., Dave, A., ... & Stoica, I. (2016). Apache Spark: A unified engine for big data processing. *Communications of the ACM, 59*(11), 56–65.