Architectural Analysis and Design for Historical NBA Sentiment Intelligence: A Data Engineering Report

Executive Summary

This report details the architectural design and feasibility analysis for a large-scale sentiment analysis pipeline targeting the r/NBA subreddit. The project scope involves the ingestion, processing, and classification of approximately two million historical records derived from archival Reddit dumps, utilizing Anthropic's Claude 3 Haiku for natural language processing (NLP) and Amazon Web Services (AWS) for storage and analytics.

The investigation into the 2024-2025 status of Reddit data accessibility confirms that the "Arctic Shift" and updated Pushshift archives, distributed via Academic Torrents, constitute the primary viable data source. These archives utilize ZStandard (ZST) compression with high window sizes, necessitating specialized Python processing pipelines to mitigate memory allocation errors during decompression.

For the core sentiment classification task, a comparative analysis of Large Language Models (LLMs) identifies the **Claude 3 Haiku (Legacy)** and **Claude 3.5 Haiku** models as the most cost-efficient solutions, particularly when leveraged through Anthropic's Message Batches API. This approach offers a 50% cost reduction compared to standard synchronous API calls, making bulk processing of 2 million records financially viable for independent data engineering projects.

The architectural review of AWS storage options concludes that a serverless data lake model utilizing **Amazon S3** (Standard Storage) and **Amazon Athena** (Serverless SQL) significantly outperforms traditional database solutions like Amazon DynamoDB or Amazon RDS in terms of cost-efficiency and analytical flexibility for this specific workload. While DynamoDB excels in high-velocity transactional writes, its cost structure for bulk ingestion and full-table analytical scans renders it economically inefficient for historical sentiment aggregation.

The proposed architecture integrates streaming ZST extraction on Amazon EC2 Spot Instances, batch LLM processing, Parquet-optimized storage on S3, and SQL-based analytics via Athena. This design ensures scalability, minimizes operational overhead, and maintains a projected infrastructure cost of less than $5.00 per month post-ingestion, with the primary expenditure being the one-time LLM inference fee.

1. The Historical Reddit Data Ecosystem (2024-2025)

The landscape of social media data preservation has undergone a paradigm shift following the restricted access to Reddit's official API in mid-2023. For data engineers and researchers, the "live" API is no longer a feasible ingress point for historical bulk data due to prohibitive rate limits and cost structures. Consequently, the community has migrated toward decentralized, peer-to-peer distribution models for archival data.

1.1 Academic Torrents and the Decentralized Archive

The primary repository for historical Reddit data is now **Academic Torrents**, a distributed file-sharing system designed to host large scientific datasets. Following the cessation of the public Pushshift API's historical endpoints, the "Arctic Shift" project and legacy Pushshift maintainers have adopted this platform to distribute monthly archive dumps.[1]

These dumps function as immutable snapshots of Reddit's database. The current availability status for the 2024-2025 period indicates a robust, albeit slightly delayed, pipeline of data releases:

- **Monthly Dumps:** Data is aggregated and released in monthly batches (e.g., RC_2024-01.zst for comments, RS_2024-01.zst for submissions). This cadence ensures that researchers have access to nearly complete datasets, usually with a lag of one to two months to account for ingestion and verification processes.[1]
- **Per-Subreddit Availability:** While the maintainers occasionally release pre-split dumps organized by subreddit (e.g., a single file containing all r/NBA comments), these are generated less frequently than the bulk monthly files due to the immense computational resources required to sort and split petabytes of data.[1] For the 2024-2025 window, relying solely on per-subreddit dumps is risky due to potential availability gaps. The recommended strategy is to ingest the bulk monthly files and filter for subreddit: nba during the extraction phase.
- **Completeness and Reliability:** The coverage of these dumps is estimated to exceed 99% of public Reddit activity.[1] However, discrepancies exist regarding deleted content. Unlike the live API, which reflects the current state of a post (potentially [deleted]), archive dumps capture the state of the content at the moment of ingestion. This enables the analysis of content that may have been subsequently removed, providing a richer dataset for sentiment analysis, though it raises ethical considerations regarding user privacy that must be managed during data cleaning.

1.2 The ZStandard (ZST) Compression Paradigm

To manage the exorbitant storage requirements of Reddit's textual history—which spans billions of comments—the archives utilize **ZStandard (zstd)** compression. Developed by Meta, ZStandard offers a superior balance of compression ratio and decompression speed compared to legacy formats like Gzip or Bzip2.[3]

The implications of this format for data engineering are profound:

- **Compression Efficiency:** The dumps achieve compression ratios often exceeding 7:1. A

file size of 10GB on disk can expand to over 70GB of raw JSON text.[4] This expansion factor necessitates a streaming architecture, as loading entire uncompressed files into Random Access Memory (RAM) is infeasible for standard workstations or cloud instances.
- **NDJSON Structure:** The internal data structure is Newline Delimited JSON (NDJSON), where each line represents a distinct, valid JSON object. This structure is ideal for streaming, allowing processors to read, parse, and discard data line-by-line without parsing the entire file structure.[5]

1.3 Data Volume Analysis for r/NBA

To accurately size the architecture, we must quantify the "2 million records" requirement against the actual activity of the r/NBA subreddit. r/NBA is consistently among the most active communities on the platform, particularly during live game threads and the off-season "F5 season."

- **Activity Metrics:** Historical analysis suggests that r/NBA generates millions of comments annually. For instance, during high-engagement events like the NBA Finals or the trade deadline, daily comment volumes can surge into the tens of thousands.[6]
- **Volume Estimation:**
  - **Peak Season (Playoffs/Finals):** Daily volumes can exceed 50,000 comments.
  - **Regular Season:** Daily averages often range between 15,000 to 30,000 comments.
  - **Off-Season:** Activity remains substantial but lower, typically 5,000 to 10,000 comments per day.
- **Scope of 2 Million Records:** Based on these metrics, a dataset of 2 million records represents approximately:
  - **~1.5 to 2 months** of regular-season activity.
  - **~1 month** of high-intensity playoff activity.
  - ~4 to 6 months of off-season discussion.
    This confirms that 2 million records is a representative sample for sentiment analysis, large enough to capture trends but small enough to remain cost-effective for proof-of-concept architectures.

2. Advanced Data Engineering: The Extraction Pipeline

The processing of Reddit ZST dumps presents specific technical challenges that standard extraction libraries often fail to address. The primary obstacle lies in the compression parameters used during the creation of the archives.

2.1 The "Max Window Size" Constraint

The Pushshift and Arctic Shift archives were compressed using aggressive ZStandard settings to minimize file size. Crucially, they utilize a large **Window Size**—the buffer used to reference back to previous data patterns for compression.

- **The Technical Failure:** Standard Python libraries (like zstandard or even command-line

tools) typically default to a window size limit of 128MB or similar to prevent excessive memory usage. However, the Reddit dumps often require a window size of **2GB (2^31 bytes)**.[7] Attempting to decompress these files without explicit configuration results in a zstd.ZstdError: Frame requires too much memory for decoding.[9]
- **The Architectural Solution:** The extraction pipeline must utilize the zstandard library with a custom dctx (decompression context) configured with max_window_size=2147483648. This explicitly authorizes the allocator to reserve the necessary memory buffer.[8]

2.2 Streaming Architecture Design

Given the disparate sizes of compressed vs. uncompressed data, a "load-and-process" approach is structurally invalid. The pipeline must be designed as a **Stream Processor**.

Proposed Python Implementation Pattern:

1. **Binary Stream Initialization:** Open the source .zst file in binary read mode (rb).
2. **Context Configuration:** Initialize the ZstdDecompressor with the mandated 2GB window size.
3. **Stream Wrapping:** Wrap the file handle in a stream_reader. This object creates a buffered interface that reads compressed chunks from the disk and yields decompressed bytes.
4. **Text Decoding:** Wrap the byte stream in io.TextIOWrapper (specifying utf-8). This layer handles the conversion of raw bytes into string data, managing multi-byte characters that might be split across chunk boundaries.
5. **Line-by-Line Iteration:** Iterate through the text stream. Each iteration yields a single line of JSON.
6. **Just-in-Time Filtering:** Parse the line into a dictionary. Immediately check the subreddit field.
   - If subreddit.lower() == 'nba', retain the record.
   - If not, discard the record immediately. This ensures that the memory footprint of the application remains constant (effectively the size of one line + buffer), regardless of the total file size.[4]

2.3 Benchmarking and Performance

Processing throughput is heavily dependent on the efficiency of the JSON parser and the filter logic.

- **Single-Core Constraints:** Python's Global Interpreter Lock (GIL) limits CPU-bound tasks like JSON parsing to a single core. For extremely large monthly dumps (e.g., 30GB compressed), a single-core script may take 12-24 hours to process.[12]
- **Multiprocessing Strategy:** To accelerate extraction, a multiprocessing architecture (e.g., combine_folder_multiprocess.py from u/Watchful1) is recommended. This approach spawns multiple worker processes, each handling a specific chunk of the file or different

files in a directory, significantly saturating the I/O bandwidth and CPU cores of the processing instance.[12]

3. Sentiment Intelligence: The Claude Haiku Architecture

For the classification layer, the selection of the Large Language Model (LLM) is governed by a trade-off between semantic nuance and unit cost. Anthropic's Claude 3 Haiku family has emerged as the optimal solution for high-volume tasks.

3.1 Model Comparison: Claude 3 vs. Claude 3.5 Haiku

The architecture must choose between two distinct versions of the Haiku model, each with different cost-performance profiles.

- **Claude 3 Haiku (Legacy):**
  - **Cost Profile:** Extremely economical. Input costs are **$0.25 per million tokens (MTok)** and output costs are **$1.25/MTok**.[13]
  - **Capabilities:** Fast and efficient. It excels at basic classification tasks but may struggle with deep contextual nuance, such as detecting the layered sarcasm common in sports subreddits (e.g., "LeBron is totally washed" posted after a 40-point game).[15]
- **Claude 3.5 Haiku (New):**
  - **Cost Profile:** Higher tier. Input costs are **$0.80 - $1.00/MTok** and output costs are **$4.00 - $5.00/MTok**.[14]
  - **Capabilities:** This model represents a significant leap in intelligence, outperforming the previous flagship (Claude 3 Opus) on many benchmarks.[17] It offers superior instruction following and nuance detection, making it better suited for accurately interpreting the "NBA Twitter/Reddit" vernacular.[15]

Strategic Recommendation: For a dataset of 2 million records, the total cost difference is absolute but manageable (approx. $30 vs. $100). If the project goal is high-fidelity sentiment trends (separating genuine criticism from memes), **Claude 3.5 Haiku** is the recommended choice. If the goal is purely directional trend analysis (e.g., "win" = positive), the legacy model suffices.

3.2 The Batch API Advantage

The single most critical factor for cost optimization in this architecture is the utilization of Anthropic's **Message Batches API**.

- **Mechanism:** The Batch API allows users to upload a file of up to 10,000 requests (or 100,000 depending on tier) for asynchronous processing. The API processes these requests within a 24-hour Service Level Agreement (SLA) window.[19]

- **Economic Impact:** Requests processed via the Batch API receive a **50% discount** on standard pricing. This fundamentally alters the economics of the project, bringing the cost of advanced NLP down to the level of basic keyword scraping.
  - *Standard Input:* $0.25 (Haiku 3) / $0.80 (Haiku 3.5)
  - *Batch Input:* **$0.125 (Haiku 3) / $0.40 (Haiku 3.5)**
  - *Batch Output:* **$0.625 (Haiku 3) / $2.00 (Haiku 3.5)**.[14]

3.3 Prompt Engineering and Caching

While Anthropic offers **Prompt Caching** (caching system prompts to reduce input costs), its applicability to this specific architecture is nuanced.

- **Thresholds:** Caching is effective for long context windows (>1024 tokens). However, a typical sentiment analysis system prompt is short ("Classify this text as Positive, Negative, or Neutral...").
- **Cost-Benefit:** Writing to the cache incurs a surcharge (e.g., $0.30/MTok for Haiku 3 writes). Unless the prompt is exceptionally long, the overhead of cache writes combined with the ephemeral nature of the cache (5-minute TTL by default) may not yield net savings compared to the flat 50% discount of the Batch API.[19]
- **Conclusion:** The **Batch API alone** is the primary driver of savings. Prompt caching should only be implemented if the instructions require providing extensive few-shot examples (e.g., 50+ examples of NBA slang) that exceed 1,000 tokens per request.

3.4 Financial Modeling for 2 Million Records

We assume a standard data profile for Reddit comments:

- **Input Size:** 100 tokens per record (50 tokens of comment text + 50 tokens of system instructions).
- **Output Size:** 5 tokens per record (e.g., "POSITIVE", JSON output).
- **Total Records:** 2,000,000.

Total Token Volume:

- Input: $2,000,000 \times 100 = 200,000,000$ (200 MTok)
- Output: $2,000,000 \times 5 = 10,000,000$ (10 MTok)

Scenario A: Claude 3 Haiku (Legacy) via Batch API

- Input Cost: $200 \text{ MTok} \times \$0.125 = \$25.00$
- Output Cost: $10 \text{ MTok} \times \$0.625 = \$6.25$
- **Total Inference Cost: $31.25**

Scenario B: Claude 3.5 Haiku (New) via Batch API

- Input Cost: $200 \text{ MTok} \times \$0.40 = \$80.00$
- Output Cost: $10 \text{ MTok} \times \$2.00 = \$20.00$

- **Total Inference Cost: $100.00**

This modeling demonstrates that using state-of-the-art LLMs for millions of records is no longer the domain of enterprise budgets. A cost of $100 for high-fidelity sentiment analysis on a massive dataset represents exceptional value.

4. AWS Cloud Infrastructure: Storage and Analytics

Selecting the appropriate storage engine is the definitive factor in the long-term operational cost and performance of the data pipeline. We evaluate Amazon S3, DynamoDB, RDS, and Athena.

4.1 Amazon S3: The Foundation of the Data Lake

Amazon Simple Storage Service (S3) serves as the immutable storage layer for both raw ingestion and processed results.

- **Storage Classes:**
  - **S3 Standard:** Priced at **$0.023 per GB/month**. For a dataset of ~2GB (2 million compressed records), the monthly cost is negligible (~$0.05).
  - **S3 Intelligent-Tiering:** While attractive for automating cost savings, it introduces a monitoring fee of **$0.0025 per 1,000 objects**.[21] If the architecture stores each comment as a separate JSON file (2 million objects), the monitoring fee alone would be **$5.00/month**—100x the cost of the storage itself. Furthermore, objects under 128KB are ineligible for auto-tiering savings.[21]
- **Optimization:** Data should be stored as **aggregated files** (e.g., partitioned Parquet files of 100MB-500MB each) rather than individual objects. This eliminates monitoring overhead and optimizes throughput for downstream analytics.
- **Recommendation:** Use **S3 Standard**. The dataset size is too small to justify the complexity of lifecycle policies or tiering logic.

4.2 Amazon DynamoDB: The Cost Trap

DynamoDB is a high-performance NoSQL database designed for Online Transactional Processing (OLTP). While it offers single-digit millisecond latency, its pricing model is fundamentally misaligned with bulk analytical workloads.

- **Write Costs (Ingestion):**
  - DynamoDB charges per **Write Request Unit (WRU)**. One WRU allows writing 1KB.
  - Ingesting 2 million records (assuming <1KB each) requires 2 million WRUs.
  - At **$1.25 per million WRUs** (On-Demand), the ingestion cost is roughly **$2.50**.[23] This is affordable.
- **Read Costs (Analytics):**
  - The critical failure point is analysis. To calculate metrics like "Average Sentiment per Player," DynamoDB must perform a **Scan** operation, reading every item in the table.

    - A full table scan of 2GB consumes thousands of Read Capacity Units (RCUs).
    - Scanning 2 million items repeatedly for different queries becomes exponentially expensive compared to S3 scans.
- **Functional Limitations:** DynamoDB lacks native SQL aggregation functions (AVG, SUM, GROUP BY). Performing sentiment aggregation would require extracting all data to an application layer or Spark cluster, effectively defeating the purpose of using a database for the query.[25]
- **Verdict: Unsuitable.** DynamoDB introduces unnecessary cost and complexity for read-heavy analytical aggregation.

4.3 Amazon RDS: The Operational Overhead

Amazon Relational Database Service (RDS) provides familiar SQL capabilities (PostgreSQL, MySQL) but introduces server management overhead.

- **Cost Floor:** RDS instances incur an hourly compute cost regardless of usage. Even a minimal db.t3.micro instance costs approximately **$15.00 - $30.00 per month**.[27]
- **Inefficiency:** For a static historical dataset that does not require real-time transactional updates, paying for 24/7 compute capacity is financially inefficient.
- **Verdict: Overkill.** Serverless options provide the same SQL capability without the fixed monthly operational expenditure.

4.4 Amazon Athena: The Analytical Engine

Amazon Athena is a serverless interactive query service that analyzes data directly in S3 using standard SQL (based on Trino/Presto).

- **Pricing Model:** Athena charges **$5.00 per Terabyte (TB) of data scanned**.[28] There are no idle instance costs.
- **Columnar Optimization (Parquet):** By converting the sentiment data into **Apache Parquet** format, the architecture unlocks massive efficiency gains.
    - **Compression:** Parquet compresses text data significantly (often 2x-4x better than raw JSON).
    - **Columnar Scanning:** Athena only scans the specific columns requested in a query (e.g., sentiment and score), ignoring the heavy body text column.
    - **Cost Implications:** A query over 2 million records in Parquet format might scan only 50MB of data.
        - $50 \text{ MB} \approx 0.00005 \text{ TB}$
        - Cost per query: $0.00005 \times \$5.00 = \mathbf{\$0.00025}$.
- **Performance:** Athena can aggregate 2 million rows in seconds, providing the analytical power of a data warehouse at a fraction of the cost.
- **Verdict: The Superior Solution.** Athena perfectly aligns with the project's requirements: SQL capability, zero fixed costs, and high performance on static datasets.

5. Comprehensive Cost Model and Volume Estimation

5.1 Data Volume Projections

Based on the r/NBA activity metrics:

- **Raw Data Volume:** 2 million comments $\times$ ~500 bytes/comment $\approx$ **1 GB** (Uncompressed JSON).
- **Processed Data Volume:** Enriched with sentiment scores and confidence metrics $\approx$ **1.5 GB**.
- **Optimized Storage:** Converted to Snappy-compressed Parquet $\approx$ **300 MB - 500 MB** on S3.

5.2 Total Cost of Ownership (TCO) Analysis

Phase 1: One-Time Ingestion and Processing

Cost Component	Specification	Estimated Cost
Data Ingress	Download from Academic Torrents	**$0.00**
Compute (Extraction)	EC2 Spot Instance (t3.medium, 24 hrs)	**~$0.30**
LLM Inference	Claude 3.5 Haiku (Batch API)	**$100.00**
S3 API Requests	PUT Requests (uploading batches)	**< $0.01**
Total (One-Time)		**~$100.31**

Phase 2: Ongoing Monthly Operations (Storage & Analysis)

Cost Component	Specification	Monthly Cost
S3 Storage	2 GB Standard Storage ($0.023/GB)	**$0.05**

Athena Analytics	100 Analytical Queries / Month	**$0.05**
Total (Monthly)		**$0.10**

Financial Insight: The architecture is extremely capital-efficient. The operational "run rate" is effectively zero (<$1.00/month). The primary investment is the $100 allocation for high-quality intelligence via Claude 3.5 Haiku.

6. Implementation Roadmap

Step 1: Infrastructure Provisioning

Provision an **Amazon EC2 Spot Instance** (recommended: m6a.large for high network bandwidth and memory) to serve as the extraction worker. Install a headless torrent client (qbittorrent-nox) to fetch the monthly archive dumps from Academic Torrents.[5]

Step 2: The Streaming Extraction Pipeline

Develop a robust Python script to handle the ZST extraction:

- **Library:** Use zstandard with max_window_size=2147483648.
- **Logic:** Implement a stream reader that iterates line-by-line.
- **Filter:** Apply a strict filter: if json_obj['subreddit'].lower() == 'nba'.
- **Sanitization:** Remove PII and deleted content markers ([deleted]).
- **Output:** Write qualifying records to a local .jsonl buffer.

Step 3: Batch Processing with Anthropic

- **Formatting:** Convert the raw JSONL into the specific format required by Anthropic's Batch API (including custom_id matching the Reddit Comment ID).
- **Execution:** Upload the batch file and trigger the job.
- **Retrieval:** Download the batch_output.jsonl upon completion.

Step 4: Data Lake Construction

- **Transformation:** Use pandas and pyarrow to merge the sentiment results with the original metadata.
- **Serialization:** Write the dataframe to S3 as **Parquet** files, utilizing Snappy compression. Partition the data by year and month (e.g., s3://bucket/nba/year=2024/month=01/) to further optimize Athena scan costs.

Step 5: Analytical Definition

Define the schema in Amazon Athena:

SQL

```sql
CREATE EXTERNAL TABLE nba_sentiment (
  comment_id STRING,
  author STRING,
  body STRING,
  sentiment_label STRING,
  sentiment_score DOUBLE,
  created_utc TIMESTAMP
)
PARTITIONED BY (year STRING, month STRING)
STORED AS PARQUET
LOCATION 's3://my-nba-datalake/data/';
```

7. Conclusion

This research confirms that building an enterprise-grade NBA sentiment analysis pipeline is both technically feasible and economically attractive in the 2024-2025 landscape. By navigating the complexities of the **Academic Torrents** ecosystem and mastering **ZST stream processing**, data engineers can bypass the limitations of the live Reddit API.

The architectural decision to leverage **Claude 3.5 Haiku via the Batch API** unlocks high-fidelity sentiment intelligence at a 50% discount, bringing the processing cost for 2 million records to approximately $100.

Furthermore, the rejection of DynamoDB and RDS in favor of an **S3 + Athena** data lake model ensures that the project remains agile and virtually free to maintain. This serverless, decoupled architecture allows for infinite analytical flexibility without the burden of database management or fixed server costs, representing the gold standard for modern historical data analysis.

Works cited

1. Reddit comments/submissions 2005-06 to 2025-06 : r/pushshift, accessed December 13, 2025, https://www.reddit.com/r/pushshift/comments/1mcw9f5/reddit_commentssubmissions_200506_to_202506/
2. Seeking Help Accessing Reddit Data (2020–2025) on Electric Vehicles — Pushshift Down, Any Alternatives, accessed December 13, 2025,

https://www.reddit.com/r/pushshift/comments/1k19slw/seeking_help_accessing_reddit_data_20202025_on/
3. *.zst packages consistently larger than *.xz : r/archlinux - Reddit, accessed December 13, 2025, https://www.reddit.com/r/archlinux/comments/eiia99/zst_packages_consistently_larger_than_xz/
4. Help! Extract subreddit data from zst file and store it in Python : r/pushshift, accessed December 13, 2025, https://www.reddit.com/r/pushshift/comments/16bvvqq/help_extract_subreddit_data_from_zst_file_and/
5. Separate dump files for the top 40k subreddits, through the end of 2024 : r/pushshift, accessed December 13, 2025, https://www.reddit.com/r/pushshift/comments/1itme1k/separate_dump_files_for_the_top_40k_subreddits/
6. The number of times in a season that players attempted 15+ free throws in a single game. Last 10 years updated: : r/nba - Reddit, accessed December 13, 2025, https://www.reddit.com/r/nba/comments/1plg6hj/the_number_of_times_in_a_season_that_players/
7. Streaming decompression for the Reddit dumps – Wellformedness - Kyle Gorman, accessed December 13, 2025, https://www.wellformedness.com/blog/streaming-decompression-reddit-dumps/
8. Information and code examples on how to use the zstandard Python module to compress / decompress data : r/pushshift - Reddit, accessed December 13, 2025, https://www.reddit.com/r/pushshift/comments/ajmcc0/information_and_code_examples_on_how_to_use_the/
9. How can I stream a file that says "Frame requires too much memory" ? · Issue #157 · indygreg/python-zstandard - GitHub, accessed December 13, 2025, https://github.com/indygreg/python-zstandard/issues/157
10. [How to] decompress the new comment dumps with Python : r/pushshift - Reddit, accessed December 13, 2025, https://www.reddit.com/r/pushshift/comments/oa6pq9/how_to_decompress_the_new_comment_dumps_with/
11. python - How to extract .zst files into a pandas dataframe - Stack Overflow, accessed December 13, 2025, https://stackoverflow.com/questions/61067762/how-to-extract-zst-files-into-a-pandas-dataframe/61068403
12. Torrent of all dump files, plus python parsing scripts : r/pushshift - Reddit, accessed December 13, 2025, https://www.reddit.com/r/pushshift/comments/plf26g/torrent_of_all_dump_files_plus_python_parsing/
13. Claude API Pricing Calculator & Cost Guide (Dec 2025) - CostGoat, accessed December 13, 2025, https://costgoat.com/pricing/claude-api
14. Pricing - Claude Docs, accessed December 13, 2025, https://platform.claude.com/docs/en/about-claude/pricing

15. Understanding Different Claude Models: A Guide to Anthropic's AI, accessed December 13, 2025, https://teamai.com/blog/large-language-models-llms/understanding-different-claude-models/
16. Claude Pricing Explained: Subscription Plans & API Costs - IntuitionLabs, accessed December 13, 2025, https://intuitionlabs.ai/articles/claude-pricing-plans-api-costs
17. Claude 3.5 Haiku vs Claude 3 Sonnet - LLM Stats, accessed December 13, 2025, https://llm-stats.com/models/compare/claude-3-5-haiku-20241022-vs-claude-3-sonnet-20240229
18. Claude 3.5 Haiku vs. Sonnet: Speed or Power? A Comprehensive Comparison, accessed December 13, 2025, https://www.keywordsai.co/blog/claude-3-5-sonnet-vs-claude-3-5-haiku
19. Batch processing - Claude Docs, accessed December 13, 2025, https://platform.claude.com/docs/en/build-with-claude/batch-processing
20. Prompt caching - Claude Docs, accessed December 13, 2025, https://platform.claude.com/docs/en/build-with-claude/prompt-caching
21. S3 Pricing - AWS, accessed December 13, 2025, https://aws.amazon.com/s3/pricing/
22. S3 intelligent tiering costs : r/aws - Reddit, accessed December 13, 2025, https://www.reddit.com/r/aws/comments/1p67r9m/s3_intelligent_tiering_costs/
23. Amazon DynamoDB pricing for on-demand capacity - AWS, accessed December 13, 2025, https://aws.amazon.com/dynamodb/pricing/on-demand/
24. DynamoDB Pricing: A Comprehensive Guide - AWS Fundamentals, accessed December 13, 2025, https://awsfundamentals.com/blog/amazon-dynamodb-pricing-explained
25. Amazon DynamoDB Complete Guide 2025: Architecture, Pricing, Use Cases & Alternatives, accessed December 13, 2025, https://www.knowi.com/blog/amazon-dynamodb-complete-guide-2025-architecture-pricing-use-cases-alternatives/
26. DynamoDB or Aurora or RDS? : r/aws - Reddit, accessed December 13, 2025, https://www.reddit.com/r/aws/comments/1h4vox5/dynamodb_or_aurora_or_rds/
27. AWS compute price analysis - GitHub Gist, accessed December 13, 2025, https://gist.github.com/turtlemonvh/89ceb82d80bfee10c19db33f45f8a7b4
28. Amazon Athena Pricing - AWS, accessed December 13, 2025, https://aws.amazon.com/athena/pricing/
29. AWS Athena Costs in 2025: Pricing Breakdown & Optimization Tips - Cloudvisor, accessed December 13, 2025, https://cloudvisor.co/aws-athena-costs-2/