A Formal Analysis of the EDGAR EPS Parser: Architectural Patterns, Heuristic Robustness, and Strategic Evolution

Karan Vora

Trexquant Investment Interview

October 2025

Abstract

This report presents a formal analysis of the EDGAR EPS Parser, a standard-library-only Python system for extracting quarterly Earnings Per Share from SEC filings. We deconstruct its multi-modal architecture, comprising a targeted tabular search followed by a tiered narrative fallback. A detailed critique of its novel columnar selection heuristic reveals a significant over-weighting of recency signals relative to periodicity cues. We confirm and analyze a critical scoring anomaly concerning Basic vs. Diluted EPS, tracing its origin to the absence of a unified scoring philosophy across different extraction paths.

The tool is invoked as:

```
python3 parser_v4.py <input_dir> <output_csv>
```

Input files are all `.html/.htm` in `input_dir`; output is a CSV with columns `filename` and `EPS` (formatted to two decimal places). Any file with no valid EPS or that causes an error yields a blank EPS entry.

1. Architecture Overview

The extraction process is **pipeline-based**: it first attempts to parse structured tables in the HTML, and if that fails to yield a clear EPS, it falls back to free-text ("narrative") searches. The main steps are:

- **HTML Parsing:** The file's bytes are decoded and fed to our custom HTML parsers.
- **Table Phase:** We use `TableParser` (built on `html.parser.HTMLParser`) to extract all HTML tables into `Table` objects (rows: `List[List[str]]`).
  - We then call `extract_from_table(rows)` to find EPS candidates in tables.
- **Narrative Phase:** If no table-based EPS is found, we strip HTML to plain text (`Stripper`) and apply up to three regex-based text searches:
  - Strict phrase search for "earnings/income/loss per share",
  - Window-based search around literal "EPS" tokens,
  - General search for "per share" phrases.
- **Selection:** The algorithm scores and collects numeric candidates from these phases, then picks the highest-scoring value as the best EPS.

The driver (`main`) simply iterates over HTML files, calls `extract_eps_from_html`, and writes each result to the CSV. It also prints the total elapsed time.

2. Parsing and Normalization

2.1 HTML Parsing

- `TableParser` **class:** Inherits from `HTMLParser`. It accumulates all table data into structured lists of rows and cells.
  - Ignores `<script>` and `<style>` content.
  - On encountering a `<table>`, it starts a new `Table`; `<tr>` starts a new row, `<td>`/`<th>` starts a new cell.
  - Text within cells is appended and collapsed (whitespace normalized) to form each cell's string.
  - When a row ends, it's added to the current table if any cell has content. When a table ends, it's added to `self.tables` if it contains any rows.
  - **Limitation:** It does not handle `rowspan`/`colspan` specially. Complex spanning headers may not align correctly, but basic layouts work reliably.
- `Stripper` **class:** Also based on `HTMLParser`. It removes scripts, styles, and HTML tags to produce a flat plain-text string. It collapses whitespace and HTML entities, yielding continuous text used for narrative-phase regex searches.

2.2 Token Normalization

- **Number regex (`NUMBER_RE`):** Captures tokens like (`$1,234.56`), `123.45`, with optional `$`, commas, sign, parentheses.
- `normalize_num_token(tok, loss_ctx)`: Cleans a numeric token to float:
  - Strips whitespace, `$`, commas.
  - If surrounded by parentheses, treats it as negative.
  - Converts to `float`. If conversion fails, discards.
  - If `loss_ctx=True`, a positive number is forced negative (to handle phrases like "net loss of $X").
- **Loss detection:** A regex `LOSS_PHRASE_RE` searches for phrases like "loss per share" or "net loss" in labels/context.
- **Hard cap:** Any number with absolute value `>` 20 is **immediately discarded**. This is a global filter to avoid capturing non-EPS figures (e.g., NAV/share or per-share cash flows), at the cost of ignoring truly extreme EPS values.
- **Decimal vs. integer scoring:** The code rewards numbers with decimals (small EPS are likely fractional) and penalizes large integer tokens, giving a slight preference to candidates that look realistic.

3. Row and Column Filtering

3.1 Blocklist (Excluding Non-EPS Rows)

A master regex `BLOCK_ROW_RE` is used to immediately skip any row whose combined text matches common non-EPS terms. This includes:

- **Share counts:** "weighted average shares", "common shares", "shares outstanding", etc.
- **Revenue/sales/EBITDA:** Terms like "revenue", "net sales", "EBITDA", "cash flow".
- **Margins and percents:** "operating margin", "gross margin", any "%".
- **Dividends:** The word "dividends" or "cash dividends".
- **Book value/NAV:** "book value", "net asset value".

By filtering these, the parser avoids rows that mention "per share" in non-EPS contexts (e.g., dividends per share or NAV per share).

3.2 Header Detection

To make sense of tables, we first identify header-like rows:

- A row is considered **header-like** if at least 60% of its cells are non-numeric (`row_is_header_like`). We examine the first 5 rows of the table to identify the headers.

We then score columns to prefer quarterly data:

- **Column picker (`pick_best_quarter_column`):** For each column index, we concatenate that column's cells in the top few header-like rows and look for date cues:
  - **Quarter signals:** Phrases like "three months ended", "quarter ended", "3-month" add a large positive score.
  - **Annual signals:** Phrases like "year ended", "12 months", "twelve months ended" subtract points.
  - **Year recency:** We extract years (1990-2099) from the header text and add a bonus proportional to the newest year found (favoring more recent data).
  - **Left-bias:** We subtract a small amount proportional to the column index, so that if two columns score equally, the **leftmost (usually most recent quarter)** wins.
- The best-scoring column index is returned as `best_col`.
- **Header context score (`header_context_score`):** When scoring individual cells later, we also compute a per-column score by looking only at the same column across up to 3 header-like rows:

  - Similar quarter/annual cues apply (but milder weights than the global column picker).
  - Recency bonus and left bias are used as well, though smaller.
  - This assigns a contextual "header score" to each data cell based on the content of its column's header.

These mechanisms ensure the parser gravitates toward the most likely latest-quarter column, and generally avoids picking values from annual columns.

4. Table Extraction Algorithm

Once we have tables parsed and a sense of which column is likely the current quarter, the parser seeks EPS values in two phases:

4.1 EPS-Labelled Block Search

- **Find EPS rows:** Scan each row for the phrase "(earnings|income|loss) per share" (`EPS_HEADER_RE`). Each matching row is assumed to be an EPS header or label.
- **Create bands:** For each such header row at index `idx`, consider a "band" of that row plus the next few rows (up to three more). These often contain the "Basic EPS" and "Diluted EPS" lines below the header.
- **Evaluate band rows:** For each row `rr` in the band:
  - if it matches the blocklist (shares, revenue, etc.), skip it.
  - Form a **label string** from the row text (lowercased).
  - Determine **loss context:** if the header row indicated loss or the current label has "loss", set `is_loss=True`.
  - Compute a **base score** from label cues:
    * If "adjusted"/"non-GAAP"/"pro forma"/"core" appears, subtract 5 (penalize Non-GAAP).
    * If "basic" appears, add +4; if "diluted" appears, add +1 (note: currently Basic is scored higher here).
    * If neither Basic, Diluted, nor explicit EPS header is in the label, subtract a small 0.5 (less confidence in this row).
  - **Column filtering:** Only consider numeric cells in columns of interest:
    * If a `best_col` was determined, only cells in `best_col` or `best_col+1` are allowed.
    * If no `best_col` (no clear header), we skip any column whose header looks like an annual period (by scanning the header rows text).
  - **Extract numbers:** For each remaining cell `cell` in the row:

    * Ignore it if it contains a percent sign `%`.
    * Find a number token with `NUMBER_RE`.
    * Merge split parentheses if needed (`maybe_merge_paren_token`) so something like `($0.34 )` becomes `($0.34)`.
    * Normalize to a float with `normalize_num_token`, respecting loss context.
    * **Filter out** any val with |`val`| > `HARD_ABS_CAP` (20.0) to enforce the absolute cap.
    * If |`val`| >= `5` and the label had no "basic" or "diluted", skip it. This avoids taking a large number from a label that didn't explicitly say Basic/Diluted.
  - **Score each candidate:** Compute a final score for this number:

    ```
    score = base
         + header_context_score(column_index)
         + decimal_bonus(tok)
         + plausibility_penalty(val)
         + integer_penalty(tok, val)
    ```

    This combines the label base score, contextual header score, a small bonus for decimals (`+0.5` if the t
  - **Collect (score, value)** into a list of candidates.

- **Keep best from bands:** After scanning all EPS bands and rows, if we have any candidates, we select the highest-scoring one (return its value and score).

4.2 Basic/Diluted Fallback

If the above EPS-header method yields **no candidates**, we do a looser scan:

- **Scan all rows:** Look for any row whose text mentions "basic" or "diluted" (`BASIC_RE` or `DILUTED_RE`).
- For each such row (not in blocklist), we compute base scores:
  - "Adjusted/non-GAAP": -5 if present.
  - "Basic" present: +2; "Diluted": +3 (the fallback inverts the bias, slightly favoring Diluted here).
- Then we repeat the same column filtering, number extraction, and scoring as above (minus the small -0.5 for unlabeled, since label is explicitly Basic/Diluted now).
- Again, keep the highest score/value if any.

The idea is that if a table has Basic/Diluted EPS but somehow wasn't caught under the explicit "per share" header search, this pass will still catch it.

4.3 Selecting the Table Winner

From both the EPS-header pass and the Basic/Diluted fallback pass, we take all candidate (score, value) pairs, sort by score, and pick the highest one. That value is returned as the table-based EPS.

5. Narrative (Text) Extraction

- **Strict "per share" phrase (`extract_from_narrative_strict`):**
  - Scan for exact phrases like "earnings per share", "income per share", or "loss per share" (`STRICT_EPS_PHRASE_RE`).
  - For each match, look immediately after it (within 120 characters) for a number with a decimal.
  - Enforce negative sign if "loss" was in the phrase.
  - Discard if $|\text{val}| > 20$.
  - Score each candidate lightly: +0.75 if "diluted" nearby, +0.25 if "basic". A tiny fixed +0.5 is added, but basically, we pick the candidate with the highest total.
  - Return the best value found in any match. If any value was found here, we stop here.
- **EPS token window (`extract_from_text_eps_token`):**
  - Look for literal "EPS" tokens (case-insensitive).
  - For each occurrence, take a window around it (roughly 140 chars on each side).
  - Base score adjustments:
    - +2 if context includes quarter cues (e.g., "three months", "quarterly", "Q1-Q4").
    - -2 if context shows annual cues ("year ended", "six months").
    - -5 if "adjusted" is present.
    - +0.5 if "basic" is in context, +0.25 if "diluted".
  - Within the window, try to find a decimal number after the "EPS" first, else anywhere in the window.
  - Normalize it, require a decimal, discard if $|\text{val}| > 20$.
  - Final score = base + 0.5 (decimal bonus always since we require decimal) + plausibility + integer penalties.
  - Collect all, pick the best. If found, return it.
- **General "per share" (`extract_from_text`):**
  - Scan for phrases like "earnings per share" or "per basic/diluted share" (`PER_SHARE_RE`).
  - For each match, define a before/after window (∼120 chars on each side).

  - Prefer the first number **after** the phrase. If none, take the last number before the phrase. If still none, take any number in the window.
  - Then recompute context over the full window (to see if "adjusted", "basic", "diluted", "loss" appear anywhere around). Assign base: -5 for adjusted, +2 if basic, +1 if diluted. Loss context flips the sign.
  - Require decimal, cap to 20, and score like before.
  - Choose the highest score among these matches.

If any of these narrative passes finds a value, that is returned as the EPS.

6. Scoring and Heuristic Priorities

The parser's choices rely on the following guiding principles:

- **Prefer Quarterly GAAP EPS:** Quarterly cues (e.g., "three months ended", "Q1 2023") get large positive scores. Annual cues (e.g, "year ended") get strong penalties. The column-picking logic is tuned to pick the newest quarter column.
- **Penalize Non-GAAP:** Any mention of "adjusted", "non-GAAP", "pro forma", or "core" subtracts 5 points. This ensures GAAP EPS is chosen over similarly phrased adjusted figures.
- **Recency:** Newer year mentions add moderate points (`(year - 1990)/5` in header context, `/2` in column picker). This helps prefer the latest data within the table.
- **Leftmost (Recent) Bias:** When scores tie, the code slightly favors left columns (assumes many tables list the most recent quarter on the left). This is a tie-breaker, not a hard rule.
- **Decimal Bonus:** Pure decimal EPS (with a dot) gets +0.5 to discourage picking an integer value. Real EPS are often fractional.
- **Integer Penalty:** If a token has no decimal point, a penalty is applied (larger if $|val| >= 10$, smaller if $|val| < 10$). We want to avoid inadvertently capturing things like "Net income per share = 5" where a decimal might be expected.
- **Loss Normalization:** Parentheses or context "loss per share" cause a candidate to be negative. For example, `(0.35)` becomes -0.35; "loss per share: 0.35" is interpreted as -0.35. This matches financial convention.
- **Row Label Weights:**
  - In the **EPS header band** path: Basic +4, Diluted +1 (the code comments suggest the intention was the opposite, which is a known inconsistency to address).
  - In the **fallback:** Basic +2, Diluted +3.

7. Noted Inconsistencies and Considerations

Two nuanced points of contention have been identified:

- **Basic vs. Diluted Preference:** The current code weights “Basic EPS” more in the EPS-band path (Basic +4) but favors “Diluted” in the fallback path (+3 vs +2). This inconsistency means sometimes the parser might pick Basic EPS in one scenario and Diluted in another, contrary to a uniform policy. The comment in the code even contradicts itself. A recommended fix is to harmonize these weights (and clarify in comments) so that Basic or Diluted is consistently preferred across both paths.
- **Comment vs. Code Misalignment:** In the EPS band logic, a comment says “Prefer Diluted slightly over Basic”, but the numeric weights do the opposite. This should be corrected for clarity.

8. Performance and Complexity

- **Speed:** Processing one file involves HTML parsing plus regex passes. Roughly, table scanning is $\mathbf{O(tables \times rows \times cols)}$, which is cheap for typical earnings tables (tens of rows/columns). The three regex-based scans over text are linear in text length. On modern hardware, parsing and extraction for each HTML takes on the order of milliseconds to a few tens of milliseconds; the entire directory (dozens of files) runs in under a second or two.
- **Memory:** Uses lightweight data structures. Tables are lists of lists of strings. There’s no DOM, no external libraries. Memory overhead is very low (basically the HTML text plus parsed strings).
- **Robustness:** The code wraps each file’s parse in try/except, so one malformed file won’t stop the batch. It prints the total runtime at the end for logging.

9. Strengths and Improvements Over Prior Versions

This version of the parser has been a part of a continuous evolution process, learning new nuances and fixing issues with each update.

- **Very Low False Positives:** The strict blocklist and hard magnitude cap (|EPS| <= 20) aggressively eliminate non-EPS numbers. For example, it avoids capturing “book value per share = 45.12” (because 45.12 > 20) or “dividends per share = 0.50” (blocked by keyword).
- **Focused Quarter Selection:** The combination of header analysis and scoring means the parser reliably picks the latest quarter’s column even if tables mix quarterly and annual data.

- **Structured-then-Text Approach:** By trying tables first, it usually finds the exact reported EPS. The narrative fallbacks cover filings that only mention EPS in prose or have poorly formatted tables.
- **Loss and Sign Handling:** Parentheses and "loss" contexts correctly produce negative EPS, matching typical financial reports.
- **Standard Library Only:** No external dependencies makes this easy to deploy anywhere Python 3.12 runs.

10. Limitations and Future Directions

- **Rigid Heuristics:** The parser depends heavily on specific keywords and patterns. Filings with unusual wording (e.g. "Earnings attributable to shareholders per diluted share") might slip through. Regexes also need updating if terminology changes.
- **Span Handling:** Without support for HTML row/colspan, some complex table layouts might misalign data. The EPS could be in a shifted cell that the parser doesn't catch.
- **Magnitude Cap Risks:** Very high EPS values (e.g. after a stock split, EPS might exceed 20) will be dropped. If these cases are critical, one could allow an override when context strongly indicates EPS (e.g. "Q3 20XX Basic EPS" with a large number but requiring a decimal).
- **Basic vs. Diluted Policy:** As noted, the inconsistent weighting can lead to different outputs. Decide whether Basic or Diluted should win when both are present, and unify the weights.
- **Potential Overfitting to Regular Patterns:** All cues are hand-tuned. To become **more agnostic**, consider integrating NLP techniques. For example, a small language model or NLP heuristic could interpret sentences like "EPS was $0.83" without relying solely on regex windows. Or a text classification model could flag EPS mentions. Machine learning could learn from labeled filings to catch variations the regex misses.
- **No ADR/Other Share Units:** The parser does not explicitly exclude "per ADS" or foreign share references. If needed, additional patterns should be added.
- **Lack of Debug Logging:** Currently we only see final output. A verbose/debug mode could log why candidates were discarded (e.g. "skipped 5.23 because annual column," "rejected 6.78 > 20 cap") to aid tuning.

11. Recommendations for Further Enhancements

- **NLP/ML Integration:** As mentioned, exploring linguistic analysis could make the parser more flexible. For instance, Named Entity Recognition or dependency parsing might better identify which number is truly EPS when sentences are complex. A supervised approach could train on many filings to predict EPS.

- **Date Parsing:** Instead of relying on fixed phrases and left-bias, parse date strings in headers into real dates. Pick the column with the latest date. This would robustly handle situations where quarters aren't strictly left-to-right.
- **Span Support:** Improve the HTML parser to account for `rowspan`/`colspan` by replicating cells or aligning columns more accurately.
- **Adaptive Band Selection:** Sometimes "Basic"/"Diluted" lines aren't immediately below the header row. Consider expanding the band search (perhaps one row above if it contains "diluted") or scanning further downward if needed.
- **Exception Cases:** Identify and whitelist known outliers. For example, if a company's legitimate EPS exceeded 20, manually allow it or add context-specific rules.
- **Configuration Flags:** Allow tuning of parameters (caps, scores) via config file or command-line options for different use cases.

12. Conclusion

This version of parser implements a table-first, then text strategy with careful heuristics to reliably extract the latest GAAP EPS from diverse EDGAR filings. It uses HTML parsing to build table structures, scores columns for recency, and applies a layered approach to finding EPS labels. The design choices (blocklists, header scoring, sign normalization) drastically reduce errors, and the result is a robust CLI tool with only standard library dependencies.

To evolve further, focus should be on making the parsing more flexible and data-driven. Unifying the Basic/Diluted preference, adding NLP or date parsing, and improving span handling will strengthen accuracy. With these improvements, the parser will better handle edge cases and be more "agnostic" to formatting variations, achieving a comprehensive EPS extraction solution.