The Architecture of Failure: A Forensic Audit of the StatementXL Rebuild Strategy and the Case for a Cognitive Financial Engine

1. Executive Summary: The Fallacy of Deterministic Normalization

The "StatementXL Rebuild Plan" presents a vision for automating the normalization of financial data from arbitrary PDF sources into complex Excel templates. Ostensibly, the plan seeks to modernize a legacy system by shifting from hardcoded regular expressions to a "Clean-Slate" architecture comprising semantic extraction, modular services, and an "analyst-in-the-loop" review process.[1] While the strategic pivot towards semantic understanding is directionally sound, the proposed technical execution rests on a foundation of "happy path" engineering that underestimates the hostility of unstructured financial data, the computational requirements of semantic reasoning, and the cognitive limits of human operators.

The plan's core philosophy relies on a "CPU-optimized" stack utilizing deterministic libraries like pdfplumber and Camelot for extraction, openpyxl for template inference, and lightweight embeddings (all-MiniLM-L6-v2) for mapping.[1] This approach assumes that financial documents are structured enough for heuristics to work "most of the time" and that human analysts can effortlessly catch the failures. This is a fatal miscalculation. Research indicates that rule-based extraction tools fail catastrophically on the idiosyncratic layouts common in 10-Ks and private company reports.[2] Furthermore, relying on static analysis of Excel files without a calculation engine ignores the dynamic nature of financial modeling.[4]

This report serves as a rigorous, adversarial critique of the StatementXL plan. It dissects the architecture layer by layer, exposing latent risks—from the fragility of lattice-based table detection to the cognitive overload of the proposed review queue. The analysis suggests that proceeding with the current roadmap will result in a system that produces silent data corruption, frustrates users with high-latency error correction, and fails to achieve the "audit-grade" reliability required by the domain. Following the critique, comprehensive remediation strategies are provided, proposing a shift towards Vision-Language Models (VLMs), graph-based calculation engines, and dynamic, context-aware ontology expansion.

---

2. The Fragility of the "Hybrid" Extraction Pipeline

The first and most critical point of failure in the StatementXL plan lies in its data ingestion layer. The proposal advocates for a "dual-path" extraction strategy: utilizing Camelot in lattice mode for bordered tables and pdfplumber for borderless tables, with a fallback to vision APIs only when these deterministic methods flag a failure.[1] This strategy is emblematic of a misunderstanding of the topology of modern financial documents. It presumes that table structure is binary—either bordered or borderless—and that heuristic algorithms can reliably distinguish between the two without corrupting the data.

2.1 The Failure of Rule-Based Table Detection in Financial Contexts

Financial statements, particularly those found in 10-Ks, management reports, and lender presentations, rarely adhere to the strict grid structures required by lattice-based extractors. Camelot's lattice mode operates by detecting ruling lines (pixel-perfect horizontal and vertical lines) to define cell boundaries.[2] However, modern financial aesthetics often dictate the removal of vertical ruling lines, leaving whitespace as the only delimiter between columns. This renders lattice mode useless for a vast majority of the target corpus.

The plan attempts to mitigate this by falling back to pdfplumber or Camelot's "stream" mode for borderless tables. However, stream-based parsers rely on analyzing whitespace "gutters" to infer column boundaries. This heuristic approach is notoriously brittle when facing the specific typographic conventions of finance:

- **Variable Column Spacing and Spanning Headers:** Financial tables frequently employ nested column structures, such as "Three Months Ended" versus "Six Months Ended," where the super-headers span multiple sub-columns. These headers are often not perfectly centered, causing stream parsers to misidentify the column boundaries or merge distinct periods into a single data point.[3] Without a semantic understanding of the header hierarchy, a rule-based parser cannot distinguish between a spanning header and a misalignment, leading to data shifting where 2023 data is placed in the 2022 column.
- **Multi-line Rows and Text Wrapping:** A significant percentage of line items in financial statements wrap to a second line. For example, a label like "Depreciation and amortization of property, plant, and equipment" may span three visual lines. Stream parsers, which process text line-by-line, frequently fracture this single semantic row into disjoint data points or, worse, associate the second line of the label with the numerical value of the row below it.[6] This decoupling of label and value is a silent error that standard validation logic (like Assets = Liabilities + Equity) may not catch if the error affects non-balancing items or memorandum items.
- **Embedded Hierarchies and Indentation:** The visual indentation used to denote sub-totals (e.g., "Total Current Assets" being indented relative to "Cash") is often the only signal of the accounting hierarchy. pdfplumber and similar text-stream tools often flatten

this hierarchy, losing the parent-child relationships essential for downstream validation.[7]

Research comparing extraction tools on diverse document sets confirms these limitations. While rule-based tools like Camelot excel in specific, rigidly templated document types, their recall drops precipitously on heterogeneous "wild" documents compared to learning-based approaches.[2] The plan's assumption that these tools can serve as the "primary" layer is a recipe for a high volume of false negatives—tables that are missed entirely or parsed with subtle structural errors.

Remediation: Vision-First Extraction Architecture

To achieve the requisite reliability, StatementXL must abandon the rule-based waterfall in favor of a deep learning-based Table Structure Recognition (TSR) pipeline. The visual structure of a table—the spatial relationship between headers, rows, and values—is a more robust signal than the presence of ruling lines or whitespace gutters.

Table 1: Comparison of Extraction Methodologies

Feature	**Rule-Based (Camelot/pdfplumber)**	**Vision-Based (TATR/LayoutLMv3)**
Detection Mechanism	Line detection / Whitespace heuristics	Object detection / Transformer attention
Borderless Tables	High failure rate (merges columns)	High accuracy (learned spatial patterns)
Multi-line Rows	Frequently fractures rows	Context-aware row grouping
Performance	Fast on CPU (if simple)	Requires optimization for CPU latency
Robustness	Brittle to font/layout changes	Generalizes to unseen layouts

Proposed Solution:
The system should deploy Microsoft's Table Transformer (TATR) or a fine-tuned LayoutLMv3 model as the primary extraction engine.8 These models treat table detection as an object detection problem and structure recognition as a graph prediction task. By ingesting the PDF page as an image, TATR can predict the bounding boxes of rows, columns, and headers with

high precision, even in the presence of distortions, lack of borders, or complex nesting.10 To address the CPU optimization constraint mentioned in the plan, modern distilled versions of LayoutLM or quantized TATR models can run efficiently on CPU with acceptable latency (approx. 60-100ms per page), offering a superior trade-off between accuracy and speed compared to the retry-loops necessitated by heuristic failures.11 The extracted text tokens from pdfplumber or OCR should then be mapped into this predicted grid, decoupling text extraction from structure inference.12

2.2 The Limits of Tesseract and Legacy OCR

The plan's specification of Tesseract for scanned PDF processing [1] is another vestige of outdated thinking. While Tesseract is a capable engine for prose, it is fundamentally ill-suited for the sparse, high-density numerical data found in financial tables. Its LSTM-based architecture struggles to maintain columnar alignment over large whitespace gaps and frequently misinterprets critical financial punctuation.

Critique of Tesseract in this domain reveals several catastrophic failure modes:

- **Decimal Precision Errors:** Tesseract often hallucinates or drops decimal points, interpreting "1.00" as "100" or "1 00".[13] In a financial model, a magnitude error of 100x is unacceptable and can destroy the integrity of an entire dataset.
- **Column Bleed:** In dense tables where columns are separated by narrow whitespace, Tesseract often merges text from adjacent columns, conflating the year "2023" and the value "$1,000" into a single nonsense token like "2023$1,000".[14]
- **Handwriting and Annotations:** Financial documents often contain margin notes, signatures, or stamped dates. Tesseract's performance on non-standard fonts or handwriting is negligible without extensive custom training.[15]

Remediation: Layout-Aware OCR and Verification

The remediation requires a move to **Transformer-based OCR (TrOCR)** or **PaddleOCR**, which leverage pre-trained language models to correct OCR errors based on context and attend to the 2D spatial relationships of characters.[16] Furthermore, a "Redundancy Check" layer must be implemented. For critical numerical values, the system should run two disparate OCR engines (e.g., PaddleOCR and a lightweight EasyOCR model). If the results diverge, the cell is flagged for human review. This statistical validation provides a safety net that a single engine cannot.[13]

---

3. The Template Intelligence Delusion

The "Template Intelligence Engine" is pitched as the core moat of the new StatementXL, promising to map extracted data into user-provided Excel templates while preserving formulas and structure.[1] The plan proposes using the openpyxl library to parse these templates, inferring structure and "intent" via static analysis. This approach relies on a profound misunderstanding of how Excel models function in the real world.

3.1 The Myth of Inferring Intent from Static Analysis

Excel is not merely a data container; it is a Turing-complete functional programming environment. The plan assumes that "Input Cells" can be identified by heuristics such as "blue font" or "no formula".[1] This is dangerously naive for several reasons.

Visual Semantics and Heuristic Failure:
Financial models are often idiosyncratic. While "blue font for inputs" is a common convention, it is not a rule. Models frequently use cell background colors (e.g., yellow for inputs), borders, or specific named ranges to denote input fields. openpyxl's ability to extract theme-based colors and interpret conditional formatting is limited and often buggy.18 It cannot reliably render the "visual" state of a cell that a human analyst sees. For instance, a cell might appear red due to a complex conditional formatting rule based on another sheet's value; openpyxl would only see the default style, missing the semantic signal that the cell requires attention.
The Formula Evaluation Gap:
The plan proposes building a "Formula Dependency Graph" to understand the model's logic.1 However, openpyxl effectively treats formulas as text strings. It is a file reader, not a calculation engine. It cannot evaluate complex formulas, particularly those that depend on the runtime state of the data, such as INDIRECT, OFFSET, XLOOKUP, or dynamic arrays.20

- **Dependency Blindness:** If a model uses INDIRECT("A"&B1) to reference a cell, openpyxl cannot determine which cell is actually being referenced without evaluating B1. Consequently, the proposed dependency graph will be incomplete for any sophisticated model.
- **Calculated vs. Input Ambiguity:** The plan assumes that cells with formulas are "calculated" and those without are "inputs." In reality, analysts often "hardcode" formulas (e.g., typing =100+50 to show their work) or overwrite formulas with values ("plugs") during crunch time. A cell might contain a formula but *functionally* be an input for the purpose of the new period (e.g., a growth rate assumption derived from historicals that needs to be overwritten with a new projection). The static parser will misclassify these cells, leading to a failure to populate critical drivers.

Remediation: Headless Calculation and Graph Neural Networks

To truly understand an Excel template, the system must interact with it as a computational engine, not a text file.

Proposed Solution:

- **Integration of a Calculation Engine:** The system must integrate a Python-based calculation library like **xlcalculator** or **formulas**.[4] These libraries can parse and evaluate a significant subset of Excel functions, allowing the system to resolve dependencies and determine the true value and state of cells. For enterprise-grade fidelity, the backend should utilize a headless instance of **LibreOffice Calc** or a Windows worker running

Excel via COM (if SOC 2 compliance permits) to evaluate the template state *before* data insertion.

- **Semantic Graph Neural Network (GNN):** Instead of relying on brittle formatting heuristics, the system should represent the spreadsheet as a graph where nodes are cells and edges are spatial and functional relationships. A **Cell-Type Classification GNN** (e.g., based on the CCNet architecture) can be trained to predict the probability of a cell being a "Label," "Input," "Header," or "Calculation" based on its content, formula topology, and neighborhood context.[23] This probabilistic approach generalizes to new templates far better than hardcoded style rules, learning that a number surrounded by formulas is likely an input, regardless of its font color.

3.2 Semantic Alignment Flaws

The plan suggests mapping "Expected Line Items" from the template to the ontology using all-MiniLM-L6-v2.[1] This assumes that the label in the Excel cell (e.g., "Sales") is the definitive semantic key.

Critique:
In financial modeling, the semantic meaning of a cell is often defined by its context rather than its label alone. A cell labeled "Sales" under the section "Segment A" is semantically distinct from "Sales" under "Segment B." A simple embedding of the text "Sales" will collapse these distinct concepts into a single vector, causing mapping collisions.24 Furthermore, all-MiniLM-L6-v2 is a general-purpose model; its understanding of nuanced financial distinctions—such as the difference between "EBITDA," "Adjusted EBITDA," and "EBITDAR"—is superficial compared to domain-specific models.25 Benchmarks show that all-MiniLM-L6-v2 achieves significantly lower accuracy on specialized retrieval tasks compared to larger, more modern models.26
Remediation: Hierarchical Context Embedding

Proposed Solution:

- **Contextualization:** The embedding input must be constructed from the cell's full lineage. The system should concatenate > > > [Cell Label] to generate a unique semantic fingerprint for each target cell.
- **Specialized Embeddings:** The system should replace all-MiniLM-L6-v2 with **bge-m3** or a fine-tuned **FinBERT** embedding model.[26] These models support larger context windows and dense retrieval tasks, offering significantly higher separation between semantically close but financially distinct terms.[28] The marginal increase in computational cost is negligible compared to the value of accurate mapping.

---

4. The Static Ontology Bottleneck

The plan relies on a fixed "Financial Line Item Ontology" stored in YAML/JSON, containing roughly 500 core items.[1] This represents a "waterfall" approach to knowledge management that is destined to fail in the dynamic landscape of financial reporting.

4.1 The Maintenance Trap of Static Taxonomies

Financial reporting is adversarial and evolving. Companies frequently invent new "Non-GAAP" measures (e.g., "Community Adjusted EBITDA") to obfuscate performance or highlight specific business drivers. A static ontology of 500 items is hopelessly insufficient against the long tail of idiosyncratic reporting.

- **Staleness:** A static list requires manual curation. When the FASB issues a new standard or a new industry-specific metric emerges (e.g., "Daily Active Users" in tech), the static ontology becomes immediately obsolete. The plan mentions "Analyst-in-the-Loop" feedback as a learning mechanism, but without an automated way to promote new terms to the ontology, this becomes a manual bottleneck.[29]
- **Granularity Mismatch:** The ontology might contain a generic concept like "Revenue," but the document might report "Revenue from Related Parties." A simple vector match might map this to "Revenue" with high confidence, losing the "Related Party" nuance which is critical for risk analysis and covenant compliance.

4.2 The False Economy of CPU-Optimized Embeddings

The plan emphasizes CPU compatibility using all-MiniLM-L6-v2 to save costs.[1]

Critique:
While CPU inference is cheaper per compute hour, the cost of error in finance is infinite. The accuracy trade-off of using a 23M parameter model (MiniLM) versus a 300M+ parameter model (BGE-Large or E5) is significant in zero-shot classification tasks.26 In a high-stakes environment, saving $0.05 per document on compute is negligible compared to the cost of a senior analyst spending 15 minutes debugging a misclassified "Deferred Revenue" line item.
Remediation: Dynamic Knowledge Graph & RAG-Based Taxonomy

Proposed Solution:

- **Transition to Knowledge Graph:** The system must move from a static YAML list to a **Dynamic Knowledge Graph (KG)** stored in a graph database (e.g., Neo4j). The KG should model relationships (is_a, part_of, calculated_from) rather than just a flat list of terms. This allows the system to understand that "Software Revenue" is a *child* of "Total Revenue," preserving granularity.[32]
- **Retrieval-Augmented Generation (RAG):** When an extracted term does not match the core ontology, the system should use a RAG pipeline to query an external financial corpus (GAAP/IFRS guides, Investopedia) to determine if it is a synonym or a new distinct concept. An LLM agent can then propose a new node in the KG, which is provisionally added and flagged for analyst confirmation. This creates a self-healing, expanding ontology.

- **GPU Infrastructure:** Acknowledge that high-accuracy semantic mapping requires GPU acceleration. Deploy the embedding service on GPU-enabled serverless containers (e.g., AWS Lambda with Provisioned Concurrency or NVIDIA Triton Inference Server). The latency improvement (sub-10ms vs 50ms+) and accuracy gains justify the marginal infrastructure cost increase.[34]

---

5. Human-in-the-Loop (HITL) Fatigue & UX Failure

The plan envisions a "Review Queue" where analysts resolve ambiguities, sorted by impact.[1] While logical on paper, this design fails to account for human cognitive limitations and the specific workflow of financial review.

5.1 The "Review Queue" Cognitive Overload

- **Alert Fatigue:** If the extraction layer (Camelot) and mapping layer (MiniLM) perform at their typical baseline for complex documents, the "Ambiguity Rate" will likely exceed 30-40%. If an analyst has to manually confirm or correct 50 items per document, they will quickly develop "click-through" behavior, approving suggestions without reading them just to clear the queue.[36] This "rubber stamping" defeats the purpose of the audit trail and compromises data integrity.
- **Context Switching:** The proposed UI shows a "PDF Snippet" next to the cell.[1] This is insufficient. Financial validation requires broad context (e.g., "Does this number match the MD&A narrative on page 45?" or "Is this consistent with the footnote on revenue recognition?"). Confining the user to a snippet view deprives them of the document-level context needed for judgment.[38]
- **The Fallacy of Prioritization:** The plan prioritizes high-impact items like Revenue. However, the *real* pain points in financial data extraction are the obscure, low-impact items (e.g., "Other comprehensive income (loss), net of tax") that cause the balance sheet to fail to balance. By focusing on the easy, high-value items, the system leaves the analyst to hunt for the "needle in the haystack" errors that break the accounting equation.

Remediation: Active Learning & Balance-Sheet Validation

Proposed Solution:

- **"Balance-Sheet-First" Validation:** The UI should not present a flat list of ambiguities. It should present the **Accounting Equation**. The system should calculate Assets - (Liabilities + Equity) and highlight the *difference* (e.g., "Off by $1.2M"). By clicking the difference, the system effectively performs a "Goal Seek," using the embedding engine to find unmapped items in the PDF that sum to approximately $1.2M. This changes the task from "data entry" to "puzzle solving," leveraging the analyst's domain intuition and reducing cognitive load.[38]

- **Synced Document View:** The Review UI must support "synced scrolling," where clicking a cell in the Excel grid auto-scrolls the full PDF view to the exact source page and draws a bounding box around the value. This preserves the document context necessary for verification.[39]
- **Active Learning Loop:** The system must implement an active learning loop where analyst corrections immediately retrain a lightweight adapter (LoRA) for the classification model. This ensures the system creates a "personal" or "entity-specific" profile that rapidly stops making the same mistakes, reducing the review burden over time.[40]

---

6. Technical Stack & Infrastructure Vulnerabilities

6.1 Python/FastAPI Scalability Bottlenecks

The plan selects Python (FastAPI) for the backend.[1] While Python is the lingua franca of AI, it is notoriously inefficient for CPU-bound tasks like parsing massive XML (Excel) files or iterating through PDF objects.

- **GIL Limitations:** Python's Global Interpreter Lock (GIL) will choke the "Extraction Service" if it attempts to process multiple large PDFs concurrently on a single worker, even with async constructs.[34]
- **Memory Overhead:** openpyxl is memory-inefficient. Loading a complex, formula-heavy LBO model can consume gigabytes of RAM, potentially causing OOM (Out of Memory) kills in a containerized environment.[42]

Remediation: Polyglot Architecture

Proposed Solution:

- **Rust/Go for Parsing:** Offload the heavy lifting of PDF parsing and Excel DOM manipulation to a high-performance language like **Rust** (using libraries like calamine for Excel reading and pdf-rs for parsing). These services can be exposed as Python bindings (via PyO3) or microservices. This can result in 10-100x speedups for file I/O operations and eliminate the GIL bottleneck.[44]
- **Async Worker Pools:** Use robust task orchestration systems like **Celery** or **Temporal**. Long-running extraction jobs must be decoupled from the API request/response cycle to prevent timeouts and ensure scalability.[34]

6.2 Security & Multi-Tenancy Risks

The plan treats security as a "Phase 8" concern.[1] In the context of financial data, which often contains Material Non-Public Information (MNPI), this is a critical oversight.

- **Data Leakage:** Using a shared embedding model or RAG vector store for multiple tenants creates a risk of cross-contamination. If Company A's "Project X" is embedded, there is a non-zero risk that Company B's semantic query could inadvertently retrieve it if filters fail.

- **Model Poisoning:** If the system learns from analyst overrides globally, a malicious or incompetent user could map "Revenue" to "Net Income" repeatedly. If the global model learns this bad mapping, it could corrupt the inference for other clients.

Remediation: Tenant-Isolated Vector Spaces

Proposed Solution:

- **Row-Level Security (RLS):** Implement PostgreSQL RLS immediately (Phase 1) for all data primitives.
- **Namespaced Vector Stores:** Ensure the vector database (e.g., Qdrant, Pinecone) uses strict tenant namespacing. No query should ever execute without a mandatory tenant filter.
- **Federated Adaptation:** Do not train a single global model on user feedback. Train **Tenant-Specific LoRA Adapters**. This ensures that custom mappings (e.g., "Company A treats 'Marketing' as 'COGS'") remain isolated to that tenant and do not pollute the global model.[40]

---

7. Comprehensive Reconstruction: The "Anti-Fragile" Architecture

Based on the critiques above, the following architectural blueprint is proposed to replace the "Clean-Slate" plan. This architecture prioritizes robustness, semantic understanding, and security.

7.1 The "Sovereign" Data Pipeline

Core Shift: From "Extract -> Map" to "Understand -> Reason -> Generate."

Layer	Component	Replacement for	Rationale
Ingestion	**Rust-Based Parser** (pdf-rs, calamine)	Python pdfplumber / openpyxl	Provides 10x throughput, memory safety, and eliminates GIL blocking for heavy I/O operations.
Extraction	**LayoutLMv3 / TATR** (Fine-tuned)	Camelot / Tesseract	Handles borderless tables, complex layouts, and OCR

			correction via visual context, avoiding heuristic failures.
Intelligence	**Graph Neural Network (GNN)**	Heuristic Rules (Blue font)	Probabilistically detects input/output cells based on graph topology and formula dependencies rather than brittle formatting.
Mapping	**BGE-M3** + **Neo4j KG**	MiniLM + Static YAML	Delivers deep semantic understanding with large context windows and a dynamic, self-expanding ontology.
Validation	**Agentic Solver** (LangGraph)	Greedy Algorithm	Uses an agentic workflow to "backtrack" and solve the Accounting Equation, treating validation as a constraint satisfaction problem.

7.2 Implementation Roadmap Adjustments

To realize this architecture, the implementation roadmap must be fundamentally altered:

1. **Phase 1 (Immediate):** Implement the **Vision-Grid Extractor**. Do not waste resources implementing Camelot logic branches that will inevitably be discarded.
2. **Phase 2:** Build the **Contextual Embedding Service** backed by GPU infrastructure. Skip

all-MiniLM and standardize on bge-m3 or E5-Large.
3. **Phase 3:** Develop the **Graph-Based Template Engine**. Ensure it can evaluate formulas via a headless calculation engine or robust Python library equivalents.
4. **Phase 4:** Build the **Collaborative UI**. The user experience is the product; it cannot be deferred to Phase 5.

8. Conclusion

The original StatementXL plan is a blueprint for a prototype that works on demonstration files but will drown in the edge cases of reality. It fails to account for the hostility of real-world financial data, the complexity of Excel modeling, and the cognitive needs of the human analyst. By shifting from **Heuristics + Static Lists** to **Vision Models + Dynamic Graphs**, and by upgrading the infrastructure to support the necessary compute, StatementXL can achieve the reliability required to become a defensible moat in the financial technology stack. The recommendation is to halt the execution of the original Phase 1 and re-architect the Extraction Service around Vision Transformers immediately. The path to success lies not in finding a cleaner regex, but in giving the machine the eyes to see and the brain to reason.

Works cited

1. StatementXL Rebuild Plan.txt
2. A Comparative Study of PDF Parsing Tools Across Diverse Document Categories - arXiv, accessed December 29, 2025, https://arxiv.org/html/2410.09871v1
3. Comparison with other PDF Table Extraction libraries and tools - GitHub, accessed December 29, 2025, https://github.com/camelot-dev/camelot/wiki/Comparison-with-other-PDF-Table-Extraction-libraries-and-tools
4. xlcalculator - PyPI, accessed December 29, 2025, https://pypi.org/project/xlcalculator/
5. pycel - PyPI, accessed December 29, 2025, https://pypi.org/project/pycel/
6. tabula vs camelot for table extraction from PDF - Stack Overflow, accessed December 29, 2025, https://stackoverflow.com/questions/61387304/tabula-vs-camelot-for-table-extraction-from-pdf
7. Heuristic Algorithm for Automatic Extraction Relational Data from Spreadsheet Hierarchical Tables - The Science and Information (SAI) Organization, accessed December 29, 2025, https://thesai.org/Downloads/Volume12No10/Paper_82-Heuristic_Algorithm_for_Automatic_Extraction_Relational_Data.pdf
8. Benchmarking Extraction of Structured Data from Templatized Documents - UC Berkeley EECS, accessed December 29, 2025, https://www2.eecs.berkeley.edu/Pubs/TechRpts/2025/EECS-2025-77.pdf
9. Table Transformer (TATR) is a deep learning model for extracting tables from unstructured documents (PDFs and images). This is also the official repository for

the PubTables-1M dataset and GriTS evaluation metric. - GitHub, accessed December 29, 2025, https://github.com/microsoft/table-transformer
10. Table Detection and Transformation Using TATR (Table Transform... - E2E Networks, accessed December 29, 2025, https://www.e2enetworks.com/blog/table-detection-and-transformation-using-tatr-table-transformer-using-tatr-on-e2e-cloud
11. PP-DocLayout: A Unified Document Layout Detection Model to Accelerate Large-Scale Data Construction - arXiv, accessed December 29, 2025, https://arxiv.org/html/2503.17213v1
12. Structure PDF Table Data for AI Applications with GMFT - codesphere - Ghost, accessed December 29, 2025, https://codesphere.ghost.io/ai-table-extraction-gmft/
13. Extraction of data from table is not accurate · Issue #2081 · docling-project/docling - GitHub, accessed December 29, 2025, https://github.com/docling-project/docling/issues/2081
14. Financial Table Extraction in Image Documents - arXiv, accessed December 29, 2025, https://arxiv.org/html/2405.05260
15. A Guide to Optical Character Recognition (OCR) With Tesseract - Unstract, accessed December 29, 2025, https://unstract.com/blog/guide-to-optical-character-recognition-with-tesseract-ocr/
16. A template-independent approach for information extraction in real estate documents - CEUR-WS.org, accessed December 29, 2025, https://ceur-ws.org/Vol-3486/17.pdf
17. Fine-tune LayoutLMv3 with Your Custom Data | by It's Amit | Medium, accessed December 29, 2025, https://mr-amit.medium.com/fine-tune-layoutlmv3-with-your-custom-data-7435f6069677
18. Full support for reading font and fill colors · Issue #116 · weshatheleopard/rubyXL - GitHub, accessed December 29, 2025, https://github.com/weshatheleopard/rubyXL/issues/116
19. Excel Automation with Python: Conditional Formatting in Excel with Python - Cursa, accessed December 29, 2025, https://cursa.app/en/page/excel-automation-with-python-conditional-formatting-in-excel-with-python
20. python 3.x - openpyxl - Limitation of DataValidation - Stack Overflow, accessed December 29, 2025, https://stackoverflow.com/questions/76088507/openpyxl-limitation-of-datavalidation
21. The OFFSET function cannot use the return value of the INDIRECT function. · Issue #4403 · PHPOffice/PhpSpreadsheet - GitHub, accessed December 29, 2025, https://github.com/PHPOffice/PhpSpreadsheet/issues/4403
22. formulas Documentation, accessed December 29, 2025, https://formulas.readthedocs.io/_/downloads/en/stable/pdf/
23. Semantic Structure Extraction for Spreadsheet Tables with a Multi-task Learning

Architecture - Microsoft, accessed December 29, 2025, https://www.microsoft.com/en-us/research/wp-content/uploads/2019/12/TableStructure_DI19.pdf
24. [P] Benchmarking Semantic vs. Lexical Deduplication on the Banking77 Dataset. Result: 50.4% redundancy found using Vector Embeddings (all-MiniLM-L6-v2). : r/MachineLearning - Reddit, accessed December 29, 2025, https://www.reddit.com/r/MachineLearning/comments/1prjy85/p_benchmarking_semantic_vs_lexical_deduplication/
25. FinMTEB: Finance Massive Text Embedding Benchmark - arXiv, accessed December 29, 2025, https://arxiv.org/html/2502.10990v2
26. Benchmark of 11 Best Open Source Embedding Models for RAG - Research AIMultiple, accessed December 29, 2025, https://research.aimultiple.com/open-source-embedding-models/
27. Financial Sentiment Analysis and Classification: A Comparative Study of Fine-Tuned Deep Learning Models - MDPI, accessed December 29, 2025, https://www.mdpi.com/2227-7072/13/2/75
28. Reducing False Positives in Retrieval-Augmented Generation (RAG) Semantic Caching: a Banking Case Study - InfoQ, accessed December 29, 2025, https://www.infoq.com/articles/reducing-false-positives-retrieval-augmented-generation/
29. Challenges of using LLMs for reasoning over complex ontologies | by Gio Ortiz - Medium, accessed December 29, 2025, https://gio-ortiz.medium.com/challenges-of-using-llms-for-reasoning-over-complex-ontologies-43feeca8ac89
30. A Short Review for Ontology Learning: Stride to Large Language Models Trend - arXiv, accessed December 29, 2025, https://arxiv.org/html/2404.14991v2
31. Best Open-Source Embedding Models Benchmarked and Ranked - Supermemory, accessed December 29, 2025, https://supermemory.ai/blog/best-open-source-embedding-models-benchmarked-and-ranked/
32. Insights, Techniques, and Evaluation for LLM-Driven Knowledge Graphs | NVIDIA Technical Blog, accessed December 29, 2025, https://developer.nvidia.com/blog/insights-techniques-and-evaluation-for-llm-driven-knowledge-graphs/
33. Knowledge Graph and LLM Integration: Benefits & Challenges - FalkorDB, accessed December 29, 2025, https://www.falkordb.com/blog/knowledge-graph-llm/
34. How can you incorporate Sentence Transformers in a real-time application where new sentences arrive continuously (streaming inference of embeddings)? - Zilliz Vector Database, accessed December 29, 2025, https://zilliz.com/ai-faq/how-can-you-incorporate-sentence-transformers-in-a-realtime-application-where-new-sentences-arrive-continuously-streaming-inference-of-embeddings
35. How can you improve the inference speed of Sentence Transformer models, especially when encoding large batches of sentences? - Milvus, accessed

December 29, 2025, https://milvus.io/ai-quick-reference/how-can-you-improve-the-inference-speed-of-sentence-transformer-models-especially-when-encoding-large-batches-of-sentences

36. Cognitive effects of prolonged continuous human-machine interaction: The case for mental state-based adaptive interfaces - PubMed Central, accessed December 29, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10790890/
37. The Myth of the Human-in-the-Loop and the Reality of Cognitive Offloading - Perry World House, accessed December 29, 2025, https://perryworldhouse.upenn.edu/news-and-insight/the-myth-of-the-human-in-the-loop-and-the-reality-of-cognitive-offloading/
38. The impact of AI errors in a human-in-the-loop process - PMC - NIH, accessed December 29, 2025, https://pmc.ncbi.nlm.nih.gov/articles/PMC10772030/
39. A Guide to Excel Spreadsheets in Python With openpyxl, accessed December 29, 2025, https://realpython.com/openpyxl-excel-spreadsheets-python/
40. Graph-boosted Active Learning for Multi-Source Entity Resolution - Uni Mannheim, accessed December 29, 2025, https://www.uni-mannheim.de/media/Einrichtungen/dws/Files_Research/Web-based_Systems/pub/Primpeli-Bizer-ALMSER-ISWC2021-Preprint.pdf
41. Informativeness-Based Active Learning for Entity Resolution, accessed December 29, 2025, https://old.dbs.uni-leipzig.de/file/DINA2019_paper_5.pdf
42. OpenPYXL Poor Performance Optimisation - The Null Route, accessed December 29, 2025, https://blog.dchidell.com/2019/06/24/openpyxl-poor-performance-optimisation/
43. Parsing Irregular Spreadsheet Tables in Humanitarian Datasets (with Some Help from GPT-3) | by Matthew Harris | TDS Archive | Medium, accessed December 29, 2025, https://medium.com/data-science/parsing-irregular-spreadsheet-tables-in-humanitarian-datasets-with-some-help-from-gpt-3-57efb3d80d45
44. Fastest way to process 1 TB worth of pdf data : r/dataengineering - Reddit, accessed December 29, 2025, https://www.reddit.com/r/dataengineering/comments/1ioewek/fastest_way_to_process_1_tb_worth_of_pdf_data/