Architectural recommendations for financial document processing mostly fail small teams

Most commonly recommended architectures for financial document AI are over-engineered for small teams. After investigating practitioner experiences, GitHub issues, and production case studies, the evidence reveals a stark gap between what research papers propose and what actually works. For a 1-2 person team building multi-tenant SaaS processing **10,000 documents/day**, the optimal stack is dramatically simpler than typically recommended: Python + Celery, pdfplumber/PyMuPDF, PostgreSQL with recursive CTEs, and bge-base embeddings—no VLMs, no GNNs, no graph databases, no Rust.

Vision-language models are not production-ready for table extraction

The most striking finding is the **complete absence of documented enterprise deployments** of LayoutLMv3, TATR, or Table Transformer for financial document processing. Despite extensive searching across GitHub issues, production case studies, and practitioner forums, zero large-scale production deployments of open-source VLMs were found. Companies processing documents at scale universally use managed services (AWS Textract, Google Document AI, Azure Document Intelligence), not self-deployed open-source VLMs.

The fine-tuning burden is severe. LayoutLMv3 requires **500-2,000 annotated documents** minimum, with each document needing bounding box and entity annotations. One GitHub thread reported 2-3 months of work on 4× A100 GPUs just for pretraining. (GitHub) The model's **512-token limit** creates significant complexity for multi-page documents—requiring chunking strategies, per-chunk inference, and result merging. (GitHub)

Inference costs run $0.01-0.05 per page on GPU infrastructure with throughput of only 10-50 pages/minute. Common failure modes from GitHub issues include CUDA out-of-memory errors even with small batch sizes, unstable header detection in Table Transformer (Issue #115), (GitHub) and tight cropping requirements where images with significant padding cause the model to fail. (Hugging Face)

Simpler alternatives achieve 80% of the accuracy. A benchmark across 800+ documents found: (arXiv)

- PyMuPDF processes documents in **0.1 seconds** (Medium) (github) (vs. ~1-10 seconds for VLMs)
- Camelot achieves **99%+ accuracy** on tables with clear gridlines
- pdfplumber reaches **96% accuracy** on structured tables with visual debugging (ACM Other conferences)
- Rule-based tools outperform TATR on government tenders, manuals, and mixed documents (arXiv)

The practical recommendation: use pdfplumber + Camelot for 80% of use cases. Reserve VLMs only for scanned scientific documents without gridlines, and even then consider managed services over self-deployment.

GNN-based spreadsheet classification appears to be academic fiction

The research found **no evidence** that "CCNet" for spreadsheet cell classification exists as a production system. Searches returned only unrelated models: Criss-Cross Network (image segmentation), ccNet (RNA sequencing), and classifier chains for general ML. If this recommendation originated from a research paper, it appears to reference either a misremembered name or an internal model never released.

More broadly, GNN approaches for spreadsheet understanding exist only in academic papers with no production deployments found. Microsoft's TableSense (CNN-based, AAAI 2019) is the closest to production but remains research-only. The fundamental blocker is training data: the DECO dataset, the largest annotated spreadsheet corpus, contains only **1,165 files**—three to four orders of magnitude too small for robust GNN training.

Production financial systems take entirely different approaches:

- **Bloomberg** uses function-based queries (BDP, BDH, BDS) delivering pre-structured data Dartmouth College
- **FactSet** uses Databricks Lakehouse with SQL and Pandas Medium
- **S&P Capital IQ** uses SQL queries and Excel plugins with SQL formulas
- **Visible Alpha** relies on analyst model structure that's already semi-structured G2

What actually works for financial Excel processing: **openpyxl for reading, position + formatting heuristics for header detection,** MIT Press **regex for content classification, and dictionary lookup for term mapping**. Published algorithms like HUSS and RAC achieve good accuracy with simple rules plus error correction. Semantic Scholar

Embedding model selection is usually wrong

The research uncovered several surprising findings that challenge conventional wisdom about embedding models for financial text.

all-MiniLM-L6-v2 performs poorly despite popularity. Despite 200M+ HuggingFace downloads, it achieved only **56% Top-5 accuracy** in RAG benchmarks—among the lowest scores tested. AIMultiple Its 2019 architecture cannot compete with modern retrieval-optimized models like e5-base (100% accuracy, same speed tier).

FinBERT is NOT an embedding model—it's a sentiment classifier. Hugging Face On the FinMTEB benchmark (EMNLP 2025), FinBERT scored only **0.110 on retrieval tasks** versus 0.646 for bgelarge-en-v1.5. Using FinBERT for semantic search is a category error.

A shocking finding: **Bag-of-Words outperforms all dense embeddings on financial semantic text similarity** arXiv (0.485 vs. maximum 0.434 for dense models). The explanation: extensive boilerplate content in annual reports introduces noise, and 27% unique financial terms per document reduces lexical overlap that dense models learn to exploit. arxiv

Domain adaptation provides **modest 4-15% gains**, less than commonly claimed. arxiv arXiv Fine-tuning on your own data typically produces larger improvements than switching to pre-trained financial models. Production financial firms (AlphaSense, Bloomberg, Kensho) universally use **custom models trained on proprietary corpora**, not off-the-shelf sentence transformers. Dailybaileyai

Practical recommendation: Start with bge-base-en-v1.5 or e5-base (~30ms latency, runs on CPU). Implement **hybrid retrieval (dense + BM25)** to capture the lexical matching that dense models miss on financial text. supermemory Skip MiniLM and FinBERT entirely.

Python handles 10,000 documents per day without breaking a sweat

The claim that Python is "too slow" for document processing at SaaS scale is comprehensively false. **PyMuPDF processes documents in 0.1 seconds average**, Medium github translating to 36,000 documents per hour on a single thread. A calculation for 10,000 documents/day:

- PyMuPDF at 0.1s/doc = **17 minutes total processing time**
- With 4 Celery workers = approximately 4 minutes
- A single t3.large EC2 instance (~$60/month) provides 345,600 documents/day capacity
- 10,000 documents/day represents **3% of capacity**

Production evidence supports this. AWS's GenAI IDP Accelerator powers Ricoh processing 70,000 healthcare documents/month and Competiscan processing 35,000-45,000 marketing documents daily — AWS both using Python-based architectures. AWS explicitly states their solutions have "processed millions of pages in a day."

The actual bottleneck hierarchy in document processing:

1. **OCR** (1-30 seconds per page for scanned documents)
2. **LLM inference** (2-10 seconds per document for GPT-4 calls)
3. **Network I/O** (uploading/downloading documents)
4. **Database operations**
5. **PDF parsing** (<5% of total pipeline time)

None of these bottlenecks are solved by switching from Python to Rust/Go. The cost-benefit analysis is decisive: Rust implementation requires ~160 hours minimum at $100/hour = $16,000 in opportunity cost. Savings in compute might be $54/month. Break-even: approximately 25 years.

Rust/Go becomes justified only at **500,000+ documents/day** sustained, when infrastructure costs exceed $10,000/month, or when sub-100ms latency is required. For a 1-2 person team, polyglot architecture is unequivocally premature optimization.

Graph databases are unnecessary for financial taxonomies

For a 500-2,000 item financial line item ontology, Neo4j is almost certainly unnecessary. More importantly, **no evidence was found that Bloomberg, FactSet, S&P Capital IQ, or Refinitiv use graph databases for their taxonomies**. All use SQL-centric architectures or proprietary data platforms:

- Bloomberg: Unix servers with C++/Fortran, proprietary databases
- FactSet: Databricks Lakehouse with Delta tables and SQL Medium
- S&P Capital IQ: Databricks with SQL queries, won awards for "Virtual Database Platform" S&P Global
- XBRL taxonomies (10,000+ elements): Stored as XML Schema, not graph databases

The "self-healing ontology with RAG" pattern is **academic fantasy with no production evidence**. The only implementations found were demos using Harry Potter books, not financial systems. One practitioner wrote: "After processing a thousand clinical reports, my 'knowledge graph' was a mess of duplicates, inconsistencies, and missing data." Medium

Financial taxonomies require stability for regulatory compliance. Self-healing implies unpredictable changes—antithetical to audit requirements. XBRL taxonomies are updated annually by standards bodies, not automatically.

A PostgreSQL solution handles 500-2,000 items trivially: PostgreSQL

```
sql
```

```
WITH RECURSIVE category_hierarchy AS (
  SELECT id, name, parent_id, 0 AS depth
  FROM line_items WHERE parent_id IS NULL
  UNION ALL
  SELECT c.id, c.name, c.parent_id, h.depth + 1
  FROM line_items c
  INNER JOIN category_hierarchy h ON c.parent_id = h.id
)
SELECT * FROM category_hierarchy;
```

For 500-2,000 items (~1MB), even in-memory dictionaries work perfectly with microsecond traversal. Graph databases become justified only at millions of entities with complex many-to-many relationships and graph analytics requirements (PageRank, community detection).

What production document systems actually use

Cloud providers use multi-model architectures, not single end-to-end systems:

- **AWS Textract**: CNN-based layout analysis + specialized models per document type Cloudchipr
- **Azure Document Intelligence**: CNNs for visual features + RNNs for sequences + NLP for context Medium
- **Google Document AI**: Vertex AI foundation models tuned for documents, 25 years of OCR research Google Cloud

Open-source alternatives have different architectures:

- **Unstructured.io**: Python ETL pipeline using Tesseract OCR, poppler-utils, LibreOffice. GitHub Their proprietary "Chipper" model uses transformer-based Visual Document Understanding IARPA
- **Docling (IBM)**: Two core models—layout analysis (CNN-based, trained on 81,000 labeled pages) and TableFormer for table extraction. Procycons New Granite-Docling-258M is an ultra-compact VLM at only 258M parameters

Successful startups follow patterns:

- **Rossum** ($100M Series A): Proprietary transactional LLM trained on millions of documents, Rossum continuous learning from human feedback Rossum
- **Eigen Technologies** (acquired by Sirion): Small data AI requiring <500 training documents, probabilistic graphical models over neural networks, Deep Analysis used by ~40% of G-SIBs

- **Hyperscience**: Model-first approach with strong handwritten extraction, ~$1.50/page (Opticintellect)

The real bottleneck is mapping accuracy, not extraction

Production evidence clearly shows the actual bottleneck ranking:

1. **Mapping accuracy (primary bottleneck)**: Getting extracted values into correct target schema fields. Research shows multi-stage pipelines with page retrieval achieve **8.8× higher field-level accuracy** versus direct VLM processing. Long documents with mixed content exceed context windows, causing degraded extraction. (arXiv)
2. **Human review UX (close second)**: HITL systems reduce costs by up to 70%, but poorly designed interfaces create productivity bottlenecks. The EasyData case study showed reduction from **7.5 minutes to 7.5 seconds per document** with optimized review interface. (EasyData)
3. **Extraction accuracy (largely solved)**: Modern OCR achieves 95%+ accuracy on clean documents. This is no longer the limiting factor.

Proven approaches to reduce human review burden:

- **Confidence-based routing**: Only flag extractions below 90% confidence (Parseur)
- **Active learning**: Every human correction feeds back into training—Rossum claims "AI learns with every human keystroke" (Rossum)
- **Staged review reduction**: Start with 100% review of borderline cases, reduce to 25% sampling after two stable weeks (Beetroot)
- **Multi-stage pipelines**: Pre-processing → OCR → Page classification → Targeted extraction reduces errors and improves interpretability (arXiv)

Practical architecture for a 1-2 person team

Based on the evidence, the recommended stack for building multi-tenant financial document processing SaaS:

Processing layer: Python + Celery with Redis broker. PyMuPDF for fast extraction, (Medium) pdfplumber for tables. (pythonology) Start with 2-4 workers, scale to 8-16 as needed.

ML/AI layer: bge-base-en-v1.5 for embeddings (runs on CPU, 30ms latency). Hybrid retrieval with BM25. For complex extraction, use OpenAI/Claude API calls rather than self-hosted VLMs.

Data layer: PostgreSQL with recursive CTEs for taxonomy hierarchy, (Neon) JSONB for flexible metadata. Skip Neo4j entirely.

Document processing approach: Rule-based extraction first (pdfplumber, Camelot). Cloud APIs (Textract, Google Document AI) for complex scanned documents. LLM calls for semantic field mapping.

Critical investment areas: Build excellent human review UX from day one. Track straight-through processing rate as primary KPI. Implement confidence scoring on all fields.

What to avoid: Self-deployed VLMs (LayoutLMv3, Table Transformer), GNN-based spreadsheet classification, graph databases for taxonomies, Rust/Go rewrites, all-MiniLM-L6-v2, FinBERT for embeddings, and "self-healing ontologies with RAG."

Conclusion

The gap between academic recommendations and production reality is substantial. Most architectures commonly proposed for financial document processing are designed for organizations with dedicated ML teams, GPU infrastructure, and months of training data collection—not 1-2 person SaaS teams.

The irony is that simpler approaches often perform better: pdfplumber beats Table Transformer on many document types, [arxiv] [arXiv] BM25 + dense hybrid retrieval beats pure dense embeddings on financial text, and PostgreSQL handles taxonomies that don't need Neo4j's complexity. The 80/20 rule applies aggressively—80% of the value comes from well-executed simple solutions.

The real architectural insight is that **document processing is a UX problem as much as an ML problem**. The bottleneck isn't extraction accuracy—it's mapping accuracy and human review efficiency. Invest in confidence scoring, excellent review interfaces, and feedback loops before investing in more sophisticated ML models.