From 20% to 60% Precision in Six Months:

How First-Principles Thinking and Data-Centric AI Transformed Document Understanding at Adobe

Anjan Goswami, Ph.D.

Abstract

Adobe Document Cloud's AI systems—form field detection for Adobe Sign and semantic document parsing for Acrobat and Creative Cloud—had stalled at approximately 20% average precision despite years of effort involving multiple teams, complex LayoutLM-class models, and large human-annotated datasets. The problem was not the models or the data individually. It was an organizational and methodological dysfunction: a fragmented team structure where linguists, data vendors, modelers, researchers, and platform engineers operated in silos with misaligned incentives and incompatible timelines. Research produced papers with architecturally complex models that engineering could never deploy; labeling produced meticulous but glacially slow annotations; and the gap between evaluation datasets and production data was never measured, let alone closed. This case study describes how restructuring the team around a focused core unit, simplifying labeling guidelines, replacing heavyweight LayoutLM models with lightweight YOLOv5 variants, establishing production-representative test data, building a data-centric feedback loop using privacy-preserving embeddings, and implementing dark-launch validation transformed precision from 20% to 60% within six months—with acceptable latency and throughput—using fewer people than the previous effort employed. The approach generalized to multiple document understanding use cases across Adobe's product portfolio and provided the foundation for subsequent mobile deployment through model distillation.

Smartfier

1 The Problem Landscape: Multiple Use Cases, One Core Challenge

Adobe Document Cloud faced several document AI problems that shared a common technical core: **semantic parsing of visually complex documents**. Two use cases were the most critical and most stalled.

1.1 Use Case 1: Form Field Detection for Adobe Sign

When a user uploads a PDF to Adobe Sign to create a fillable form, the system must automatically detect and classify form fields. The input PDF may be a scanned document, a legacy Acrobat file, or a PDF generated in a way that stripped interactive form metadata. The system needed to:

- **Classify** whether the document contains a form (binary classifier—the first gate)
- **Detect form fields**: text inputs, checkboxes, radio buttons, signature blocks, date fields
- **Identify field types** and apply semantic constraints—names, ZIP codes, phone numbers, dates of birth—to enable input validation
- **Determine required vs. optional fields** based on visual cues (asterisks, bold labels, "required" annotations)
- **Associate footnotes with fields**: a footnote reference next to a field must link to the correct footnote text, enabling contextual help when the user fills the form

- **Detect list items** requiring tick or cross marks, with semantic inference of what each option represents

This is not a single detection problem. It is a **structured semantic parsing task**: detecting visual elements, classifying their types, establishing spatial and logical relationships between them (field-to-label, field-to-footnote, checkbox-to-option-text), and inferring constraints that are visually implied but not explicitly encoded.

1.2 Use Case 2: Semantic Document Parsing for Acrobat and Creative Cloud

Adobe Acrobat and Creative Cloud required semantic parsing of multimodal documents containing text paragraphs, images, infographics, tables, captions, headers, and footers. The system needed to:

- **Detect document elements**: paragraphs, headings, images, infographics, tables, captions, page numbers, headers, footers
- **Establish reading order**: the correct sequential flow of text across columns, around figures, through sidebars
- **Extract table structure**: row and column boundaries, merged cells, header rows, and cell contents
- **Segment visual regions**: distinguish photographs from infographics from decorative elements

This served two product surfaces. For **Acrobat**, the parsing was required to render PDF content correctly—the graphics engine needed to know which regions were text, which were images, and in what order they should be presented. For **Creative Cloud**, the parsing enabled a workflow where users could import campaign documents, advertisements, or marketing materials and edit them—but editing required converting the PDF into a structured internal format, and that conversion depended entirely on accurate semantic parsing.

1.3 The Common Technical Core

Both use cases reduce to the same underlying problem: given a document image (or PDF rendering), detect rectangular regions, classify their semantic types, and establish relationships between them. The models, training pipelines, evaluation metrics, and deployment infrastructure are shared. A solution that works for form fields should generalize, with appropriate training data, to document elements—and vice versa.

This commonality was recognized but not exploited. The two use cases were worked on by overlapping but uncoordinated teams, each making independent architectural and data decisions. The result was duplicated effort, inconsistent quality, and—critically—approximately 20% average precision across both use cases after years of investment.

2 Why Previous Efforts Failed: The Organizational Diagnosis

Upon inheriting the team, I focused the first several weeks not on models or data but on understanding **why development had been slow and results poor**. The diagnosis was organizational, not technical.

2.1 Siloed Teams Speaking Different Languages

Five distinct groups were involved in the document AI pipeline, each with different objectives, timelines, and success criteria:

Smartinfer

- **Linguists** designed labeling guidelines with meticulous precision—every edge case documented, every ambiguity resolved in advance. The guidelines ran to dozens of pages per task. Their goal was annotation consistency; their metric was inter-annotator agreement.
- **Data vendors** executed the labeling. The detailed guidelines meant each document took significant time to annotate. Adobe's culture valued thoroughness over speed, and the review process added further delay: data could not be used for training until it passed a multi-stage quality review with linguist sign-off.
- **Modelers** trained models on whatever data was available. They were disconnected from the data generation timeline and often worked with stale or insufficient datasets while waiting for the next batch.
- **Researchers** explored architecturally complex models—LayoutLM variants, multimodal transformers, graph neural networks for spatial relationships—and published papers. Their goal was state-of-the-art performance on academic benchmarks; their metric was precision/recall on curated evaluation sets.
- **Platform engineers** maintained the production inference infrastructure. They could not deploy the researchers' models because the models required GPU inference with latency and throughput characteristics that the existing platform could not support. The platform was legacy, complex, and no one was eager to modify it.

Each group was competent within its domain. The failure was at the **interfaces**: linguists optimized for annotation precision at the cost of speed; researchers optimized for model complexity at the cost of deployability; engineers couldn't deploy what researchers built; and nobody was measuring whether the evaluation data matched production data.

2.2 The Specific Dysfunctions

Labeling was too slow. Linguists had designed guidelines that micromanaged annotator decisions—specifying exactly how to handle every possible edge case in form layout. In AI, overly detailed guidelines defeat the purpose of crowdsourced annotation: you want human *judgment*, not human execution of a rigid protocol. The rigid guidelines slowed annotators, increased cost per document, and paradoxically *reduced* annotation quality because annotators spent cognitive effort on protocol compliance rather than perceptual judgment.

Research was not deployable. LayoutLM-class models achieved strong performance on academic benchmarks but required significant GPU compute, had long training times, and could not meet the latency requirements of production serving. Every iteration took days to train, making rapid experimentation impossible. No ML ops infrastructure existed—training was ad hoc, not reproducible, and not tracked.

Evaluation didn't match production. The evaluation datasets were curated collections of "representative" documents—but "representative" was defined by the linguists and researchers, not by production traffic analysis. Nobody had systematically compared the distribution of document types in evaluation data to the distribution in production. The models might have been performing well on evaluation data that bore little resemblance to what users actually uploaded.

The platform was a bottleneck, not an enabler. The inference platform was legacy code with complex deployment procedures. Engineers were risk-averse about modifications because the platform served multiple products and any change could cascade. Model format requirements (TensorRT) didn't match what modelers produced (PyTorch), creating a handoff gap at the final deployment step.

Smartifier

3 The Intervention: First Principles, Small Team, Fast Iteration

3.1 Team Restructuring

The first intervention was organizational. I extracted a small, focused core team from the existing structure: two modelers, one researcher, one strong engineer, and one data person. For the time being, I moved the existing manager aside and put my direct attention on this team along with a new manager I recruited specifically for this effort.

The team was deliberately small. Five people who communicate directly, share context, and can make decisions in real-time are faster than twenty people coordinating across organizational boundaries. The team owned the full pipeline end-to-end: data, modeling, evaluation, and deployment preparation.

I told the team: "We are starting from first principles. Everything we build will be principled, every assumption will be examined, and we will move fast."

3.2 Simplifying Labeling Guidelines

The first technical intervention was the labeling guidelines. I worked with the new manager to review the existing guidelines—dozens of pages of detailed instructions for every document type and edge case. We then worked with a senior linguist (whose approval was necessary for any guideline change) to dramatically simplify them.

The principle: **in AI annotation, you want human perceptual judgment, not human execution of rules.** An annotator looking at a form field should mark what they see as a form field, using their natural understanding of what a form field is. They should not be consulting a 30-page decision tree to determine whether a particular visual element qualifies under subsection 3.2.1 of the annotation protocol.

The simplified guidelines reduced annotator decision time, increased throughput, and—counterintuitively—improved consistency. When guidelines are too complex, annotators make different interpretation errors on different edge cases, producing inconsistent labels. When guidelines are simple, annotators converge on the obvious cases (which are the majority) and disagree only on genuinely ambiguous cases (which are the minority and should be handled as noise in training, not as protocol decisions).

We also set clear timelines with the data vendor and communicated that the engagement would not continue if delivery schedules were not met. This was a cultural shift: Adobe's vendor relationships were collegial and open-ended. We introduced accountability without hostility.

3.3 Replacing LayoutLM with YOLOv5

The most consequential technical decision was the model architecture. The research team had been investing in LayoutLM-class models—multimodal transformers that jointly process text, layout geometry, and visual features. These models are architecturally elegant and achieve state-of-the-art results on academic benchmarks like PubLayNet and DocBank.

They were also:

- **Slow to train**: each experiment took days, making rapid iteration impossible
- **Expensive to serve**: GPU inference with latency that exceeded production requirements
- **Difficult to deploy**: the platform engineers had no path to serving a model of this complexity within the existing infrastructure

I asked the modelers to benchmark a lightweight alternative: **YOLOv5**, a single-stage object detection model designed for speed and deployability. YOLO treats document element detection

Smartfer

as object detection—each form field, paragraph, table, or image is an object to be localized and classified.

The comparison was clear: YOLOv5 achieved approximately **10% lower precision** than LayoutLM. We validated this gap in dark launch against production traffic, and the difference was real and consistent—LayoutLM was genuinely better. This was not an evaluation artifact.

But LayoutLM could not be launched. Its GPU inference cost and latency were incompatible with the existing production infrastructure, and the precision it delivered—while better than YOLO—was still too low to justify the infrastructure investment. The business needed to see **at least 60% precision in production** to justify the GPU serving cost. The justification was concrete: Adobe Sign's marketing team was already positioning AI-powered form detection as a subscription driver. If the AI was visibly good, it would lift Sign subscriptions. If it was mediocre or required expensive infrastructure for mediocre results, the investment couldn't be defended.

YOLOv5 had decisive operational advantages:

- **Training time**: hours instead of days—enabling 5-10x more experiments per week
- **Inference latency**: acceptable for production serving on existing infrastructure without GPU
- **Model format**: YOLO's architecture is well-supported by ONNX export, enabling deployment to both server (via Triton) and mobile (via distillation)
- **Simplicity**: fewer moving parts, easier to debug, easier to understand failure modes

The strategy was explicit: take the deployable model (YOLOv5), close the precision gap through data-centric iteration, and get to 60% precision fast enough to justify a production launch. Could we have run the same data-centric loop with LayoutLM? Yes—better data improves any model. But we couldn't wait for LayoutLM's iteration cycle (days per experiment vs. hours) when the business needed a deployable system that crossed the 60% threshold within months.

3.4 Production-Representative Test Data

This was the critical missing piece. I asked the engineer to implement a **dark launch**: deploy the YOLOv5 model in shadow mode against production traffic. The model ran on real user-uploaded documents, produced predictions, but the predictions were not surfaced to users. Instead, we logged the predictions and—critically—identified the **failure cases**: documents where the model's predictions were clearly wrong.

The dark launch revealed what the curated evaluation set had hidden: the distribution of documents users actually uploaded was substantially different from the evaluation set. Production documents included scanned forms with low resolution, documents in non-English languages with different layout conventions, government forms with complex multi-column structures, and financial documents with dense tabular layouts. The evaluation set underrepresented all of these.

3.5 The Data-Centric AI Loop

With production failure cases identified, the question was how to get *similar* documents into the training set. The challenge: production documents are user-uploaded and contain private information. We could not simply add them to the training corpus.

The solution was a **privacy-preserving embedding matching** pipeline:

1. Compute document embeddings for the production failure cases using a visual encoder (operating on the rendered page image, not the text content)
2. Search a corpus of publicly available documents (government form templates, academic papers, open datasets) for documents with similar visual structure
3. Send the matched public documents to labelers for annotation

Shervin Afer

4. Add the newly annotated documents to the training set

5. Retrain, redeploy in dark launch, identify new failure cases, repeat

This created a **data-centric AI loop**: production failures drive data collection, which drives model improvement, which is validated in production, which reveals new failures. Each iteration targeted the model's actual weaknesses—not the weaknesses imagined by researchers or captured in curated benchmarks.

The loop was fast because every component was now optimized for speed: simplified labeling guidelines meant faster annotation; YOLOv5 meant faster training; dark launch meant immediate production validation.

4 Results

4.1 Precision: 20% to 60%

Within six months, average precision on production-representative evaluation data improved from approximately 20% to 60%—a 3x improvement. The improvement came from three compounding sources:

- **Better evaluation**: measuring against production-representative data instead of curated benchmarks revealed that the "20% baseline" was itself overstated on some categories and understated on others. The new evaluation was honest.
- **Better training data**: the data-centric loop targeted the model's actual failure modes with high-quality annotations of documents similar to production failures.
- **Faster iteration**: YOLOv5's short training cycle allowed 5-10x more experimental iterations, each informed by the previous dark launch.

4.2 Latency and Throughput: Acceptable and Bounded

YOLOv5 inference latency met production requirements on existing infrastructure. For throughput, we discovered through production traffic analysis that **80% of all documents uploaded to Adobe Sign were fewer than 10 pages**. By setting a threshold—the model would not process documents exceeding 10 pages—throughput remained within acceptable bounds without requiring platform optimization that engineers were unwilling to undertake on the legacy system.

This was a pragmatic engineering decision, not a modeling decision. The platform was suboptimal, but fixing it was a multi-quarter effort with high risk. Bounding the input solved the throughput problem immediately for the vast majority of users.

4.3 LayoutLM Parity Achieved Through Data, Not Architecture

The most telling result: by the end of the six-month period, YOLOv5 with the improved training data **closed the gap with LayoutLM**—achieving parity on both the original evaluation set and production traffic. The 10% gap that was real and consistent at the start was gradually erased through iterative data improvement.

This did not mean LayoutLM was unnecessary or that architecture doesn't matter. It meant that **at 20% precision, the bottleneck was data, not architecture**. Both models were starved of production-representative training data. LayoutLM's multimodal inputs (text, layout geometry) gave it a partial hedge against data gaps—it could compensate for missing visual training examples by leaning on textual and spatial features. YOLOv5, relying purely on visual features, was more sensitive to data quality—which made it more responsive to data improvements.

arXiv

The data-centric loop would have improved LayoutLM too. But the loop ran 5-10x faster with YOLO because each iteration—retrain, deploy to dark launch, identify failures, collect new data—took hours instead of days. Over six months, this compounded into dramatically more iterations, and each iteration contributed incremental precision gains. Speed of iteration, not model capacity, determined which architecture reached 60% first.

4.4 The Business Case: Justifying Production Deployment

Reaching 60% precision was not an arbitrary threshold. It was the point at which the AI form detection was **visibly useful to end users**—more than half of form fields correctly identified and pre-populated, saving users significant manual effort. Below 60%, the AI felt broken: too many missed fields meant users had to do the work anyway, and too many false positives (non-field regions marked as fields) eroded trust.

At 60%, the marketing team could credibly position AI-powered form detection as a feature that justified Adobe Sign's subscription pricing. The thesis: better AI → reduced friction in form creation → higher Sign adoption and retention → subscription lift. The 60% precision on the deployable YOLOv5 model, served on existing infrastructure without GPU cost, made this business case viable. LayoutLM at the same precision would have required GPU infrastructure investment that the subscription lift couldn't yet justify.

5 Deployment: Clearing the Last Mile

Deploying the trained model required navigating organizational processes that had historically added months of delay.

5.1 Model Format and Serving

The platform team required TensorRT format; the modeling team produced PyTorch checkpoints. I negotiated for **ONNX as the intermediate format**. ONNX served dual purposes:

- The platform team could convert ONNX to TensorRT for server deployment
- My distillation team could use the same ONNX model as a starting point for mobile deployment—YOLOv5's architecture made it amenable to model compression and distillation for on-device inference, a separate but related initiative

We containerized the model using **NVIDIA Triton Inference Server**, provided the infrastructure team with a hardware configuration sketch specifying GPU requirements and expected load patterns, and worked with them to configure the deployment.

5.2 Legal and Responsible AI Review

Adobe's deployment process required legal review (data usage compliance, IP considerations) and responsible AI certification (bias assessment, fairness review). These reviews had historically been bottlenecks because teams submitted materials reactively and iterated through feedback cycles.

We prepared the review materials proactively—submitting documentation in parallel with model development rather than after completion—and engaged the review teams early to resolve issues before they became blocking. This compressed the review timeline from the typical multi-month process to weeks.

Smartinfer

5.3 The Organizational Effect

The transformation had effects beyond the immediate precision metrics. The team—previously perceived as slow and inefficient across the organization—was suddenly delivering results at a pace that other teams found remarkable. I received unsolicited messages from colleagues saying the team was now operating like a "model team" and that they had not believed this velocity was possible, let alone with fewer people than before.

The most common question was: "What was the magic?" There was no magic. It was first-principles thinking applied systematically:

- Understand the full pipeline before optimizing any part
- Simplify where complexity adds cost but not value (labeling guidelines, model architecture)
- Measure against reality (production data), not proxies (curated benchmarks)
- Close the feedback loop between production failures and training data
- Remove organizational bottlenecks by co-locating decisions with a small, empowered team

6 Principles for Interview Discussion

6.1 Data-Centric AI Is Not a Buzzword

The data-centric approach was not ideological. It was a strategic choice driven by deployment constraints. LayoutLM was the better model—that gap was real. But LayoutLM couldn't be deployed, and a model that can't ship has zero business value regardless of its precision. We chose the deployable architecture and invested in the lever we could actually pull: data quality and iteration speed. The lesson is not "data always beats models." The lesson is: **when deployment constraints eliminate your best model, improving data on your deployable model is the fastest path to production impact.** And at 20% precision, the data improvement headroom was enormous for any architecture.

6.2 Org Design Is a Technical Decision

The team restructuring was not "management." It was a technical intervention. The siloed structure created feedback delays that made iteration cycles weeks long instead of days. Removing those delays was the single highest-leverage change. The architectural choice (YOLOv5 over LayoutLM) was the second. The data-centric loop was the third. All three were necessary; none was sufficient alone.

6.3 Lightweight Models Enable Experimentation Velocity

YOLOv5 was not chosen because it was better than LayoutLM. It was worse—consistently and measurably. It was chosen because it was **deployable and fast enough to iterate**. In a regime where data quality is the bottleneck and the business needs a production system within months, the ability to retrain in hours and validate in production daily is worth more than 10% additional precision on a model you can't ship. The precision gap closed through iteration speed; it would never have closed through model complexity alone, because LayoutLM's iteration cycle was too slow to run the data-centric loop at the pace the business required.

Smartnfer

6.4 Dark Launch Is the Most Underused Tool in ML Deployment

Running the model in shadow mode against production traffic was the intervention that changed everything. Before dark launch, the team was building models evaluated against curated data that did not represent production. After dark launch, every decision was informed by production reality. The cost of a dark launch is near zero—you're running inference but not surfacing results. The information value is enormous.

Smartinfer