Patent **R**eview **B**ot: A Comprehensive Patent Preparation and Pre-filing Tool Suite

Nicholas Zustak
nick@fenix.ai
Advisor: Dr. Bo Wu
Colorado School of Mines

August 2026

Abstract

Preparing a patent application is slow, expensive, and unforgiving of small mistakes, and the cost of an error is rarely visible until the application is already before an examiner. This project presents a patent review system that performs a structured pre-filing risk assessment of a complete application, flagging likely grounds for rejection under 35 U.S.C. §§ 101, 102, 103, and 112 before the document is filed. A central technical contribution is a document parser that reconstructs the bracketed paragraph numbers ([0001], [0002], . . .) that patent practitioners rely on to reference a specification. These numbers are generated by Microsoft Word's automatic numbering engine rather than stored as literal text, so conventional extraction tools silently discard them, and even current large language models frequently hallucinate or misattribute them when asked to cite support in a .docx file. By parsing the underlying document XML directly, the system grounds every piece of feedback in a specific, verifiable paragraph. The review tool is complemented by a small suite of patent-preparation utilities and was evaluated by practicing patent attorneys, who rated it 2.0 out of 3 on a helpfulness scale, between "very helpful" and "must-have before filing." This paper describes the system's design, the parsing problem at its core, the evaluation, and a roadmap toward agent-assisted drafting.

1 Introduction

A patent application is not free-form technical writing. It is a highly structured legal-technical document in which nearly every element serves a defined function, and in which the relationship between elements carries legal weight. A typical application contains a background section situating the invention in its field, a summary, a detailed description of the invention with reference to drawings, and, most importantly, a set of claims. The claims are the legally operative portion of the document: they define the precise boundaries of what the Applicant is entitled to exclude others from doing. Everything else in the application exists to support the claims. A useful analogy for a technical audience is a research paper in which the conclusion is legally binding, and every sentence in the body must trace back to it: the description must provide support for each term that appears in the claims, or those claims may be unsupported and vulnerable, and rejected during prosecution or even worse, invalidated after grant.

Practitioners navigate this document through its numbered paragraphs. This is a requirement at file-time from the United States Patent Office (USPTO). Each paragraph of a specification is labeled with a bracketed four-digit number, [0001], [0002], and so on, and these labels are the shared coordinate system of patent practice. An examiner's rejection will cite them, an attorney's response

will cite them, and any internal review will be phrased in terms of them. A tool that cannot refer to a specific paragraph cannot participate in this workflow in a meaningful way.

After an application is filed, a USPTO examiner reviews it and, in the overwhelming majority of cases, issues an Office Action: a formal document setting out the legal grounds on which the examiner refuses to grant the patent as filed. In my experience, over 95% of applications are rejected at the first turn (i.e., at least one claim is rejected), and only marginally more have some claims that are indicated as allowable. The ensuing back-and-forth between applicant and examiner, which includes amending claims, submitting arguments, and responding to successive Office Actions until the application is either allowed or abandoned, is called prosecution. Prosecution is governed by statute and by the Manual of Patent Examining Procedure (MPEP), the USPTO's comprehensive guide to how examiners apply the law.
Most rejections fall under one of four statutory provisions:

- **35 U.S.C. §101 (subject-matter eligibility):** whether the claimed invention is the kind of thing the patent system protects at all, or whether it is directed to an ineligible abstract idea, law of nature, or natural phenomenon. This is a particularly live issue for software and machine-learning inventions under the Supreme Court's *Alice/Mayo* framework.
- **35 U.S.C. §102 (novelty):** whether the invention is actually new, or whether it was already disclosed in a single piece of prior art.
- **35 U.S.C. §103 (non-obviousness):** whether the invention, even if not identically disclosed, would have been an obvious step for a person of ordinary skill in the field.
- **35 U.S.C. §112 (written description, enablement, definiteness):** whether the specification describes the invention fully enough that a skilled reader could make and use it, and whether the claims are supported by that description and are definite in their scope.

Each round of prosecution is expensive. It consumes attorney time, incurs USPTO fees, and can add months or years to the life of an application. There are frequently periods of up to a year where the Applicant must wait for the next Office Action. Many of the issues that surface during prosecution are, in principle, detectable before filing: a claim term that lacks antecedent basis, a specification that never adequately supports a claimed feature, or claim language that invites a subject-matter-eligibility challenge. A careful second review before filing can catch these problems while they are still cheap to fix. In practice, however, that review is time-consuming and depends on scarce senior attention, and so it is often abbreviated.

This project addresses that gap. It began from pain points in my own daily work as a patent agent preparing and prosecuting AI/ML applications, and it aims to provide an automated second set of eyes: a system that reads a complete application the way an examiner might, and surfaces the most likely grounds for rejection, grounded in specific paragraphs, before the application is filed. The remainder of this paper makes the following contributions:

- **A robust paragraph-number extraction method** that recovers the bracketed numbering conventional tools discard, by parsing Word's underlying document XML rather than its rendered text, including a hardened path for documents that have been restructured by USPTO filing systems.
- **A two-phase statutory triage architecture** that produces a fast, structured risk overview across §§ 101/102/103/112 and supports targeted, paragraph-scoped drill-down and revision suggestions on demand.

- **A design stance on feedback** in which the system offers sparse, actionable margin notes for a human drafter rather than generated replacement text, keeping a licensed practitioner in control of the document.
- **An integrated, style-matched suite** that situates the review tool alongside document-preparation utilities, and an evaluation of the review tool by practicing patent attorneys.

It is worth stating clearly at the outset that the system is a drafting aid, not a source of legal advice, and that every output is intended for review by a licensed practitioner rather than as a filing decision in itself.

2 Background and Related Work

Tools that assist patent drafting already exist, but they address different problems than the one this project targets, and the most closely related tools stop short of the paragraph-grounded, substantive review that practitioners actually want before filing.

Commercial products such as PatentBots and the proofreading features built into practice-management platforms focus largely on formal and consistency checks: claim-term antecedent basis, reference-numeral consistency between the drawings and the specification, claim-numbering integrity, and similar mechanical properties. These checks are valuable and are genuinely used in practice (I definitely use PatentBots nearly daily), but they are deliberately shallow with respect to the substantive legal questions (for example, whether a claim is likely to draw a § 101 rejection, or whether the specification adequately supports a claimed feature under § 112) that drive most of the cost of prosecution. They tell a drafter that a term lacks antecedent basis; they do not offer a view on whether the invention is claimed at an eligible level of abstraction.

At the other extreme, general-purpose large language model assistants can discuss substantive patent law fluently and will readily offer opinions on eligibility or obviousness. In practice, however, they are difficult to rely on for document-grounded review for two reasons. First, they are not reliably anchored to the specific document: asked to identify support for a claim in a specification, a general assistant will often produce plausible but unverifiable assertions, and *will frequently cite paragraph numbers that do not correspond to the actual text*, because the numbering is not reliably present in the text it was given. Second, they impose a workflow cost: they require repeated prompting and correction to stay in the required style and to focus on the right portions of a long document, which erodes the time savings they are supposed to provide.

Underlying both limitations is a technical obstacle that is specific to the file format patent work is conducted in. Patent applications are prepared and exchanged as Microsoft Word .docx files, and the bracketed paragraph numbers that practitioners depend on are not stored in those files as literal text. They are produced at display time by Word's automatic numbering engine, defined in the document's internal numbering configuration and applied to paragraphs through named styles or direct references. Every widely used text-extraction path — the popular mammoth library's raw-text extraction, python-docx's paragraph text, and ordinary copy-and-paste — returns the visible prose without these generated numbers, or corrupts them when mathematical content is present. The consequence is that any analysis built on top of conventional extraction is working from a document stripped of exactly the coordinate system it needs to give location-specific feedback. Reconstructing that numbering reliably, directly from the document's structure, is therefore a precondition for the kind of review this project sets out to provide, and it is the subject of Section 3.

3 The Paragraph-Number Extraction Problem

The feedback this system provides is only useful if it can point to specific locations in the document. A review that says "some claim lacks antecedent basis" is far less actionable than one that says "the term introduced in [0047] lacks antecedent basis in claim 3." Because patent practice is conducted entirely in terms of bracketed paragraph numbers, recovering those numbers from a .docx file reliably is a precondition for everything else the system does. This section describes why that recovery is harder than it appears, and how the system accomplishes it.

3.1 How Word generates paragraph numbers

A .docx file is not a single document but a ZIP archive of XML parts. The visible text of the document lives in word/document.xml, but the formatting that produces the rendered appearance is distributed across several other parts. Two of these are central to paragraph numbering: word/numbering.xml, which defines numbering schemes, and word/styles.xml, which defines named paragraph styles that may invoke those schemes.

Importantly, the bracketed paragraph numbers that appear in a rendered patent specification are not present as text anywhere in document.xml. They are generated at display time by Word's automatic numbering engine. A numbering definition in numbering.xml specifies a format string. For patent applications, this is a level text of [%1], where %1 is a placeholder that Word replaces with the running counter. A paragraph participates in that numbering either by carrying a style that references the definition or by referencing it directly. When Word renders the document, it walks the numbered paragraphs in order, substitutes the counter into the format string, and displays [0001], [0002], and so on. The numbers a reader sees are a rendering artifact, computed from structure, not stored content.

This design is entirely reasonable from Word's perspective — it is what allows numbering to renumber itself automatically when a paragraph is inserted — but it has a consequence that undermines every downstream tool: the numbers exist only when something performs the same rendering computation Word does.

3.2 Why conventional extraction fails

Standard text-extraction tools read the stored text of the document and do not reimplement Word's numbering engine. The mammoth library, widely used to convert .docx to plain text or HTML, extracts the textual runs from each paragraph; because the paragraph numbers are not textual runs, they are simply absent from the output. python-docx exposes each paragraph's text with the same limitation. Ordinary copy-and-paste from Word behaves inconsistently: pasting a run of same-styled paragraphs may preserve the rendered numbers, but the moment the selection includes mathematical content — even a single inline equation, which is common in machine-learning specifications — the numbering is destroyed in the pasted result.

The practical effect is that a system built on any of these extraction paths receives a specification stripped of its coordinate system. It can read the prose but cannot say where in the document that prose lives. For a tool whose entire value proposition is location-specific feedback, this is disqualifying. This is also, notably, the failure mode of general-purpose language-model assistants asked to review a pasted specification: given text in which the numbers are absent or misaligned, they will produce paragraph citations that look correct but do not correspond to the actual document, because the grounding was never present in their input.

3.3 Reconstructing the numbering by parsing document XML

The system solves this by reconstructing the numbering the same way Word does: computationally, from the document's structure. Rather than relying on extracted text, a custom parser opens the .docx archive directly in the browser using JSZip, reads document.xml, numbering.xml, and styles.xml, and reproduces the numbering computation.

The parser first resolves which numbering scheme corresponds to the patent bracket format. It scans numbering.xml for an abstract numbering definition whose level-zero format string contains the [%1] pattern, then follows the indirection from that abstract definition to the concrete numbering identifier that paragraphs actually reference. Because a paragraph may participate in numbering either through a named style or through a direct reference, the parser also collects, from styles.xml, the names of any styles that invoke the relevant numbering, so that both routes are recognized. It then walks the body of document.xml in document order; for each paragraph, it determines whether that paragraph participates in the bracket numbering, and if so, assigns it the next value of a running counter, formatted with zero-padding to match the [0001] convention. The result is a structured representation of the document in which every numbered paragraph carries its true, rendered number — the same number a practitioner would see in Word. With the numbering recovered, the parser additionally classifies the document into its structural sections — front matter, section headings, claims (with dependency relationships between claims tracked), and abstract — and assembles a full-text representation in which each paragraph is prefixed with its bracket number. It is this numbered representation, rather than a stripped text extraction, that is passed to the language model, which is what allows the model's feedback to cite real, verifiable paragraph locations.

The parser is the primary extraction path; when it cannot find bracket numbering in a document, the system falls back to mammoth text extraction, accepting the loss of paragraph-level grounding rather than failing outright.

3.4 The AS_FILED edge case

The parser as described handles documents prepared from the firm's templates, which contain a single bracket-numbering definition shared by all numbered paragraphs. In deployment, however, a large and important class of documents failed to parse: applications delivered in "as-filed" form. Every Office Action package a practitioner receives includes the application exactly as it was filed, and these documents are frequently the ones an attorney most wants to analyze — yet on these, the parser returned no numbered paragraphs and the system silently fell back to unstitched text.

Diagnosis of a representative as-filed document revealed the cause. When an application passes through USPTO filing systems or certain document-assembly pipelines, its numbering configuration is restructured. Where the original template defined one abstract numbering entry using the [%1] bracket format, the as-filed document contained 199 distinct abstract numbering entries, most of them defining the same bracket format, mapped one-to-one to 199 corresponding concrete numbering identifiers. Rather than every numbered paragraph sharing a single identifier, the paragraphs referenced a wide range of these identifiers — 229 numbering references distributed across the document, pointing at many different definitions.

The original parser assumed a single scheme: it located the first abstract definition matching the bracket pattern, resolved it to a single numbering identifier, and tested each paragraph for strict equality against that one identifier. On an as-filed document, this test matched almost no paragraphs, because the paragraphs referenced identifiers other than the single one the parser had selected. The numbered-paragraph counter therefore reached zero, and the parser reported the document as unnumbered.

The fix generalized the resolution from a single identifier to a set. Rather than selecting the first matching abstract definition and its identifier, the parser now collects every abstract definition whose level-zero format contains the bracket pattern, gathers all concrete numbering identifiers that map to any of them, and tests each paragraph for membership in that set. The change is fully backward-compatible: for a firm-template document with one bracket-numbering definition, the set contains a single element and the behavior is identical to before; for an as-filed document with 199 definitions, the set contains all of them and every numbered paragraph is correctly recognized. This generalization, together with the fallback path, is what allows the system to handle both freshly drafted applications and the as-filed documents that dominate prosecution work.

3.5 Result

The effect of the parser is most clearly seen by comparison on the same document. Passed through conventional text extraction, a patent specification emerges as continuous prose with no paragraph numbers, leaving any downstream analysis unable to cite locations Figure 1.

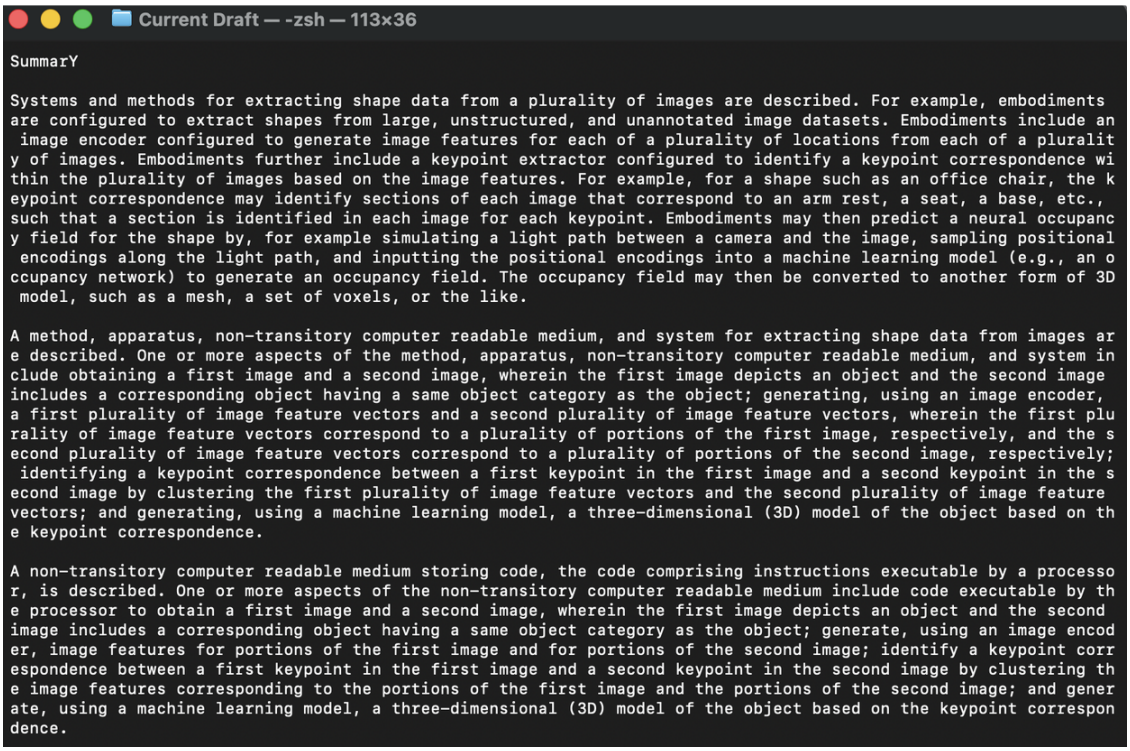


Figure 1: Output of conventional text extraction (`mammoth.extractRawText`) on a patent specification. The prose is recovered faithfully, but the bracketed paragraph numbers are absent entirely—they were never stored as text, so there is nothing for a text-based extractor to return. Any analysis built on this output has no way to cite a specific paragraph.

The prose is recovered faithfully, but the bracketed paragraph numbers are absent entirely — they were never stored as text, so there is nothing for a text-based extractor to return. Any analysis built on this output has no way to cite a specific paragraph. Passed through the structural parser, the same specification emerges with every paragraph correctly labeled, matching what the drafting attorney sees in Word Figure 2.

Each paragraph is restored to its true, rendered number (here beginning at [0003]), matching

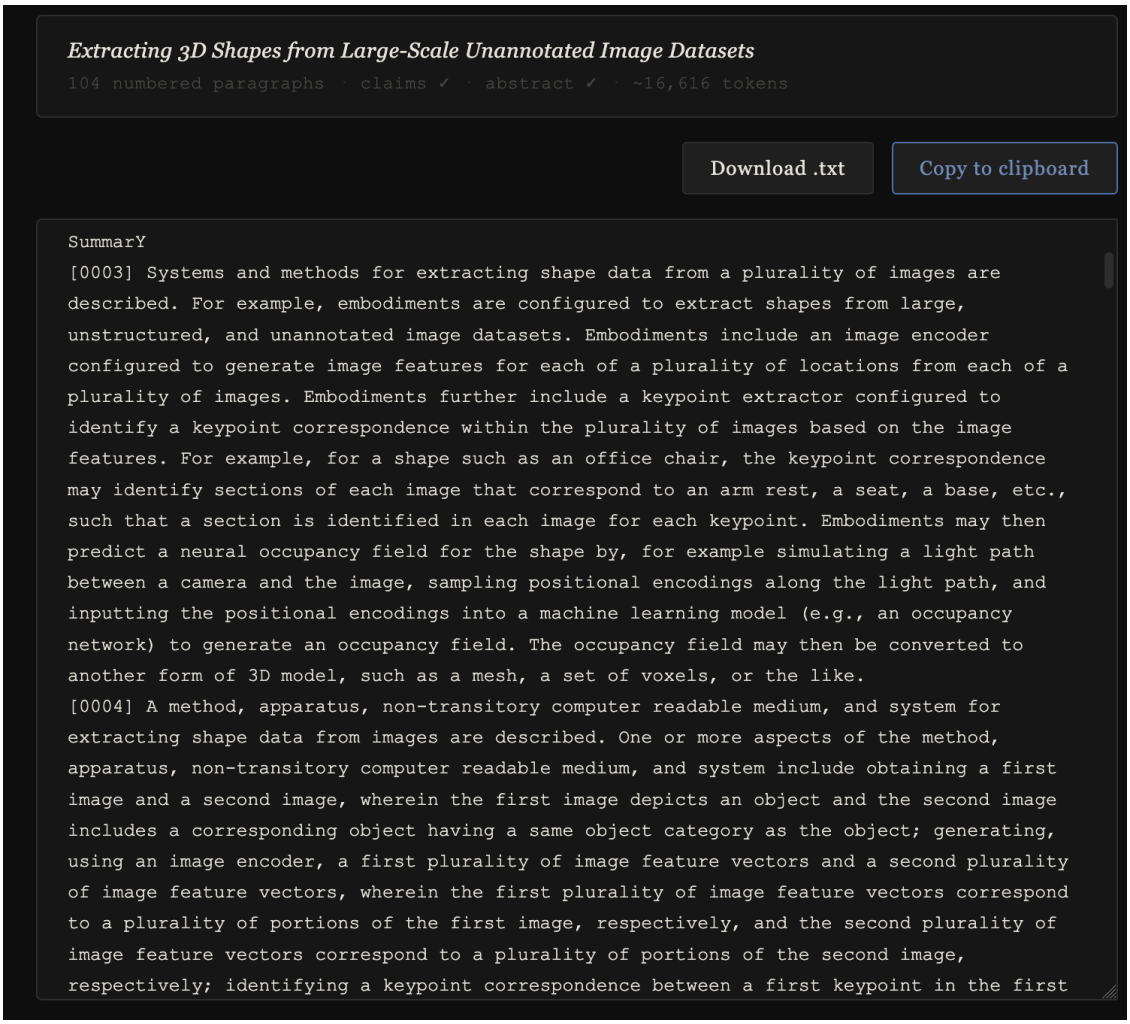


Figure 2: Output of the structural parser on the same document. Each paragraph is restored to its true, rendered number (here beginning at [0003]), matching what the drafting attorney sees in Word. The metadata bar reports 104 recovered numbered paragraphs, detected claims and abstract sections, and a token estimate—all derived from the same structural parse.

what the drafting attorney sees in Word. The metadata bar reports 104 recovered numbered paragraphs, detected claims and abstract sections, and a token estimate — all derived from the same structural parse.

The same limitation defeats the workaround a practitioner might otherwise reach for: copying text directly out of Word and pasting it into a language-model interface. Across every environment tested, pasting as plain text preserves the rendered paragraph numbers — but only until the copied range includes a mathematical object. Patent specifications for machine-learning inventions routinely express their inventive content as mathematics, encoded in the document as OMML (Office Math Markup Language). Whenever a selection includes such an object (whether an inline equation embedded in a paragraph or a displayed equation in its own table) the paragraph numbering in the pasted result is destroyed. Because a single application may contain several dozen inline equations, the practical consequence is that a practitioner cannot paste the document in one operation and retain its numbering; they are forced into a paragraph-by-paragraph copy-paste discipline that carefully avoids every mathematical region, which slows working with the model through a chat interface to a crawl. The structural parser sidesteps the problem entirely by never depending on the rendered text: it recomputes the numbering from the document's structure, and mathematical content, handled separately during parsing, does not disturb it.

Across the firm's template documents and as-filed applications used in testing, the parser recovered paragraph numbering in all documents, with any document lacking recoverable bracket numbering falling back to text extraction rather than failing. This recovered grounding is the foundation on which the statutory triage described in the next section depends.

4 System Design and Implementation

4.1 Architecture overview

The review tool is a single self-contained HTML file. It has no build step, no server-side application code, and no framework dependencies beyond two libraries loaded directly in the browser: JSZip, used by the parser to open the .docx archive, and mammoth, used only as a fallback extraction path. The entire application — document parsing, validation, prompt construction, the analysis dashboard, and all interaction logic — runs client-side in the browser. It is hosted as a static page on GitHub Pages.

This architecture was a deliberate choice given the deployment context. The tool is intended for use by practicing attorneys on firm machines, where installing software or standing up a server invites friction and IT involvement. A static page that runs entirely in the browser can be adopted by anyone with a link, and — importantly for documents as sensitive as unfiled patent applications — the .docx file itself never leaves the user's machine. Only the extracted text of the document is sent onward, and only to the language model, at the point of analysis.

A single abstraction, an LLMProvider object, isolates every detail specific to the underlying model's API — request format, streaming protocol, and token accounting. The rest of the application interacts with the model only through this object's two methods and is otherwise provider-agnostic. The practical consequence is that the analysis logic and interface are decoupled from any particular model vendor; switching providers is confined to that one object and the proxy described next.

4.2 The proxy and API-key handling

Because the application runs entirely in the browser, it cannot hold an API key directly — anything shipped to the browser is visible to its user. To keep the key confidential, requests to the model are routed through a small Cloudflare Worker that runs as a stateless proxy: it receives the browser's request, attaches the API key from an encrypted secret held server-side, forwards the request to the model provider, and streams the response back unchanged. The key is never present in the page source or transmitted to the browser. This is the only server-side component in the system, and it holds no application logic beyond forwarding.

4.3 Document validation

Before any document is sent for analysis, it passes through a rule-based validator. The validator confirms that the extracted text is long enough to be a patent application, checks for the presence of a claims section (without which substantive review is not meaningful), and detects the presence or absence of the expected structural sections — background, summary, detailed description, brief description of the drawings, and abstract. Missing required content produces a blocking error; missing optional sections produce non-blocking warnings surfaced to the user alongside a metadata summary of paragraph count and estimated token usage. The validator is a small, deterministic component, developed against a dedicated unit-test suite, and it exists so that obviously unsuitable input is caught locally and cheaply rather than consuming a model call.

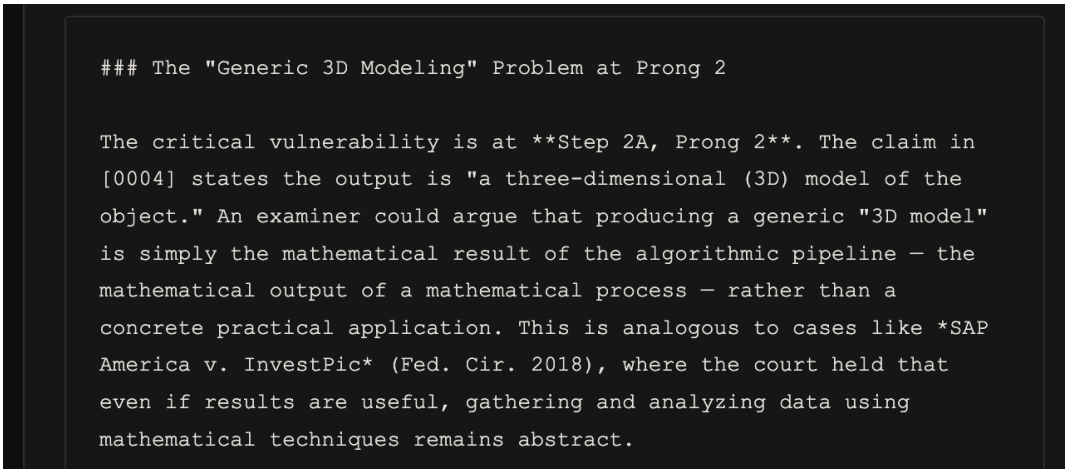
4.4 Two-phase statutory triage

The analysis itself is structured in two phases, reflecting a tension between wanting a fast, scannable overview and wanting deep analysis on demand. The first phase issues a single non-streaming request that returns a structured JSON object: the invention's title, a one-sentence summary, and — for each of the four statutory provisions (§§ 101, 102, 103, 112) — a risk level of LOW, MED, or HI, a one-sentence headline explaining the primary concern, and a list of the paragraph numbers most relevant to that concern. This structured overview is what populates the dashboard, giving the practitioner an at-a-glance read on where the application's risks concentrate. The second phase is invoked only when the user asks for more. Each medium- or high-risk finding carries a "Details" control that, on demand, issues a second request scoped narrowly to that one statutory issue — passing only the relevant paragraphs and the claims, rather than the entire document — and streams a focused analysis back into an expandable panel beneath the finding. Figure 3 shows such a drill-down for a § 101 eligibility concern: the analysis cites paragraph [0004] directly and reasons about it against controlling Federal Circuit authority (SAP America v. InvestPic), demonstrating both the paragraph-level grounding the parser makes possible and the substantive legal specificity that distinguishes this analysis from a generic summary.

4.5 The dashboard and paragraph-grounded feedback

The dashboard renders the first-phase overview as a title and summary followed by four color-coded risk cards, one per statutory provision, each showing its risk level, headline, and the set of relevant paragraph references Figure 4.

The paragraph references are the mechanism by which the system's central capability — location-specific feedback — reaches the user. Each reference is interactive. Hovering over a reference reveals a tooltip previewing the start of that paragraph's text; clicking it opens a popover containing the paragraph in full Figure 5.



```
### The "Generic 3D Modeling" Problem at Prong 2

The critical vulnerability is at **Step 2A, Prong 2**. The claim in
[0004] states the output is "a three-dimensional (3D) model of the
object." An examiner could argue that producing a generic "3D model"
is simply the mathematical result of the algorithmic pipeline — the
mathematical output of a mathematical process — rather than a
concrete practical application. This is analogous to cases like *SAP
America v. InvestPic* (Fed. Cir. 2018), where the court held that
even if results are useful, gathering and analyzing data using
mathematical techniques remains abstract.
```

Figure 3: A §101 drill-down. The streamed analysis cites paragraph `[0004]` directly and reasons about it against controlling Federal Circuit authority (*SAP America v. InvestPic*), demonstrating both the paragraph-level grounding the parser makes possible and the substantive legal specificity that distinguishes this analysis from a generic summary.

Because these references are resolved against the parsed document, in which every paragraph carries its true rendered number, a practitioner can move from a flagged risk to the exact supporting text in the specification in a single click, and verify the system's reasoning against the actual document rather than trusting an unanchored assertion. This closes the loop that conventional extraction breaks: the feedback does not merely mention a concern, it points to where in the document the concern lives.

4.6 Sparse revision suggestions

Alongside "Details," each medium- and high-risk finding offers a "Suggestions" control. This was a deliberate design decision about the form feedback should take. Rather than generating replacement claim language or rewritten specification text — which would position the tool as a ghost-drafter and invite the user to paste machine-generated text into a legal document — the suggestions are deliberately sparse: a short list of specific, actionable notes identifying what to add, clarify, or narrow and where, without drafting the text itself Figure 6.

Each note references the relevant paragraph number.

The intent is to keep a licensed practitioner in control of the document. The system functions as a senior colleague leaving margin notes, not as a substitute drafter. This suits both the professional-responsibility reality that the attorney is accountable for every word filed, and the practical observation that a well-placed prompt to a human expert is often more useful than generated text the expert must then scrutinize and rewrite.

4.7 Testing

Two components of the system are pure, deterministic logic amenable to unit testing: the document parser and the validator. Both were developed against dedicated test suites — the parser against tests covering numbering resolution, paragraph-text extraction, section classification, claim parsing, and front-matter extraction, including the multi-definition as-filed case described in Section 3.4; the validator against tests covering valid documents, missing sections, and boundary

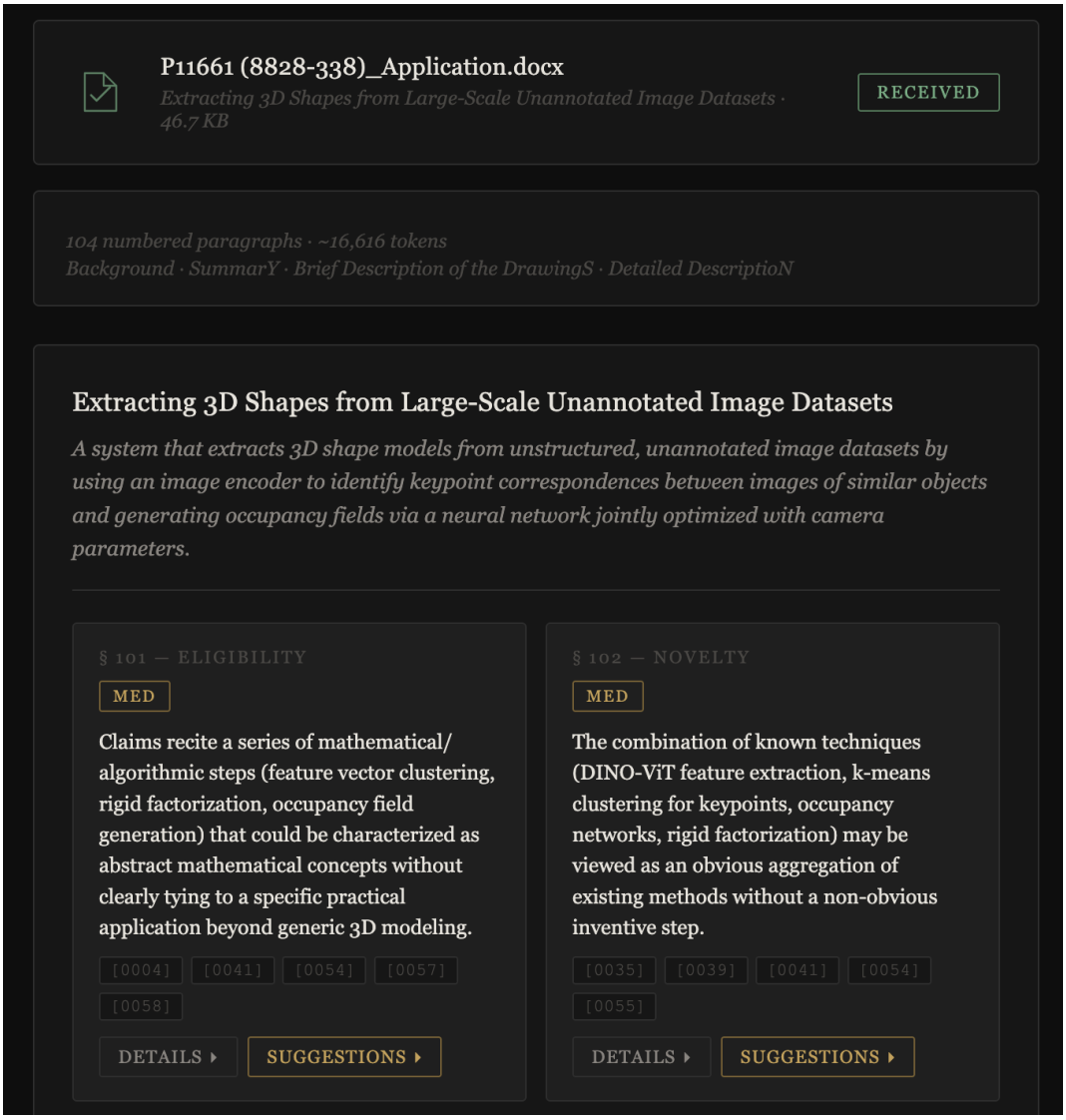


Figure 4: The at-a-glance dashboard: a file card and metadata bar, the invention title and one-sentence summary, and color-coded risk cards for §101 and §102, each showing its risk level, headline, relevant paragraph references, and the Details and Suggestions controls.

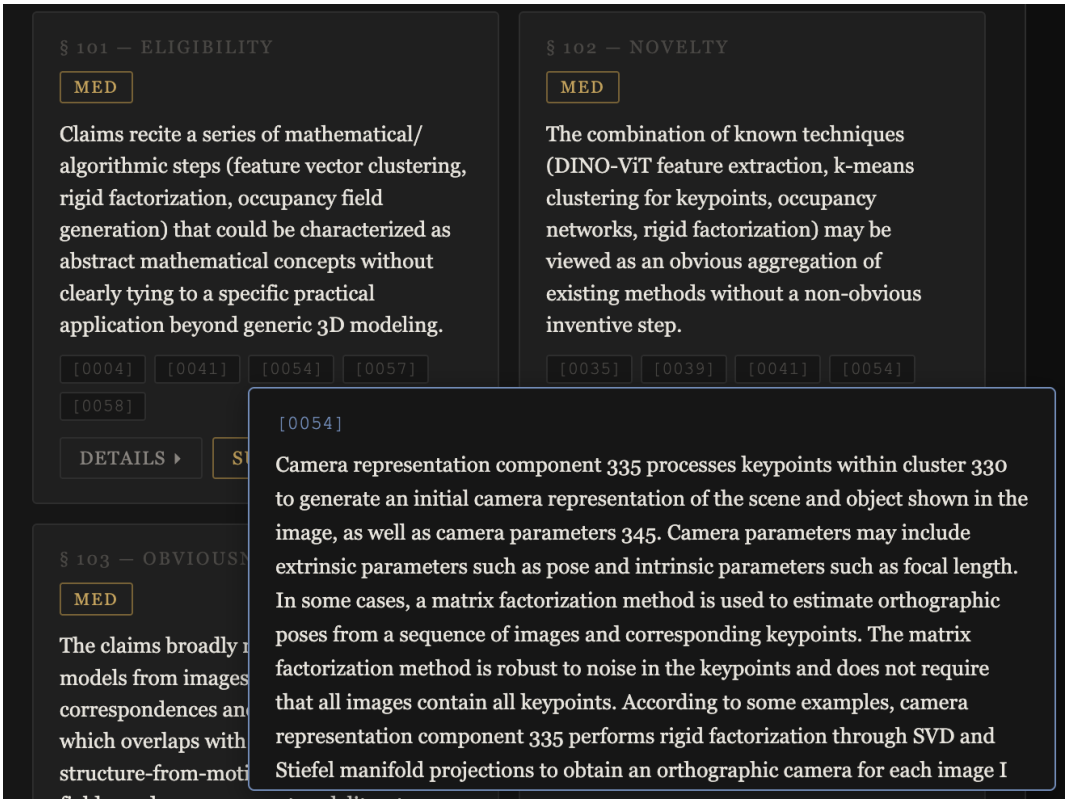


Figure 5: Clicking a paragraph reference (here `[0054]`) opens a popover containing the full paragraph text, resolved against the parsed document. A practitioner can move from a flagged risk to the exact supporting text in the specification in a single click and verify the system's reasoning against the actual document.

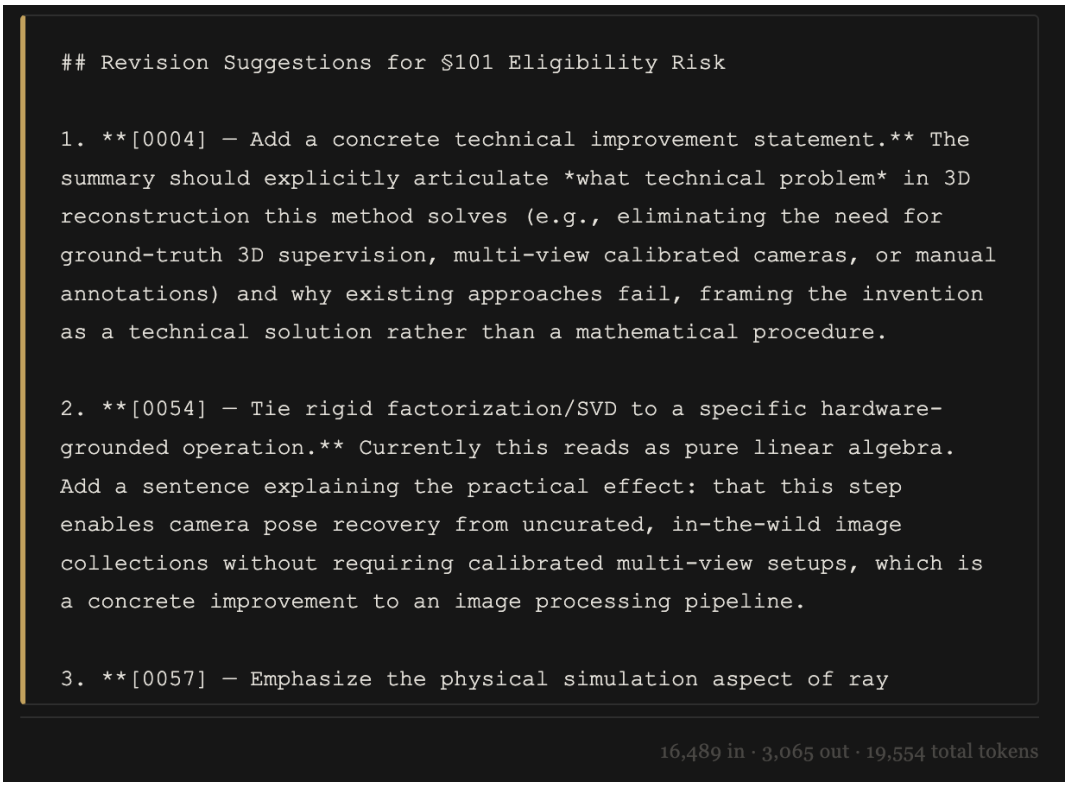


Figure 6: Revision suggestions for a §101 risk. Rather than drafting replacement text, the tool returns a short list of specific, actionable notes identifying what to add or clarify and where, each tied to a paragraph number—margin notes for a drafter rather than machine-generated claim language.

conditions. The language-model-dependent portions of the system are not unit-tested in the same way, but isolating the deterministic core behind tests means the part of the system most responsible for correctness — the recovery of paragraph grounding — is verified independently of any model behavior.

5 The Tool Suite

The review tool is the centerpiece of this project, but in practice it sits alongside two other tools that address adjacent stages of patent work. All three are presented as a single style-matched suite, reached through a shared navigation bar, and hosted together Figure 7.

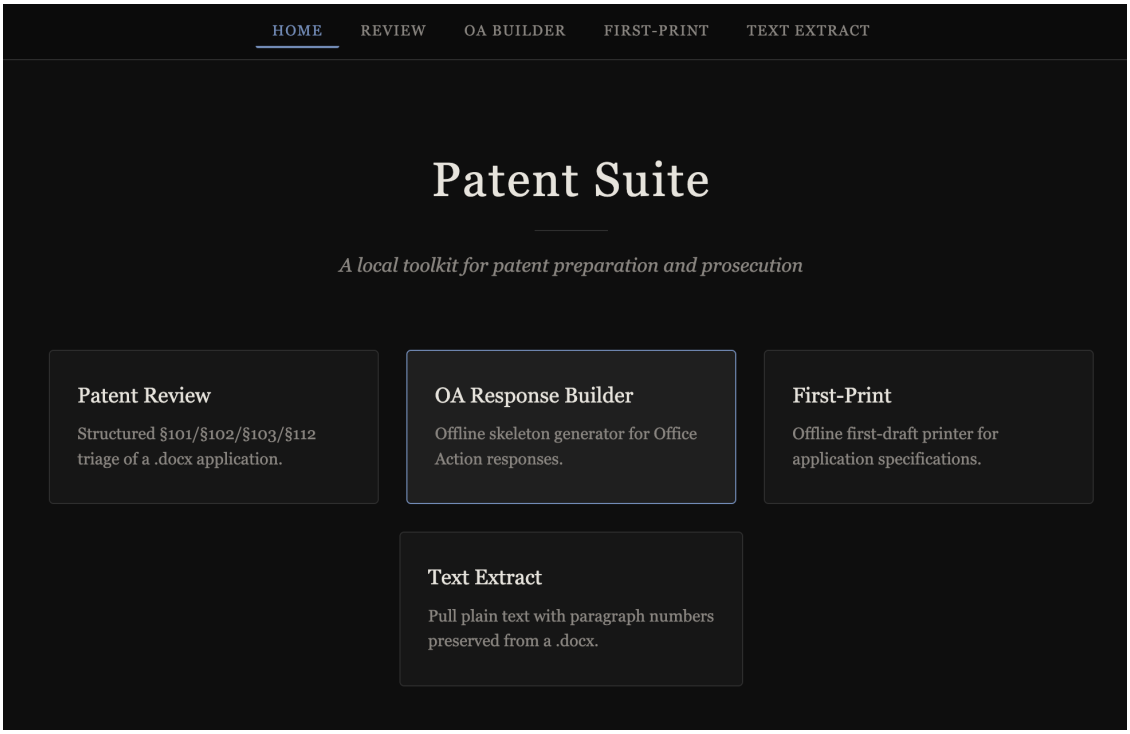


Figure 7: The suite landing page. A shared navigation bar and style-matched cards present the three independently-built tools—Patent Review, OA Response Builder, and First-Print (plus the Text Extract utility)—as one coherent application.

The suite framing reflects how the work is actually done: preparation and prosecution are a pipeline, and a practitioner moves between drafting an application, responding to Office Actions, and reviewing before filing. A design distinction runs through the suite. The review tool is inherently online — its purpose is to send document text to a language model for analysis. The other two tools are the opposite: they are entirely offline, self-contained HTML files that perform no network activity at all. This is not incidental. Both are document-generation tools that operate on live, sensitive matter — unfiled applications and draft Office Action responses — and their value proposition includes the guarantee that nothing leaves the file. Each carries a visible "runs entirely offline" indication for exactly this reason. The suite therefore deliberately combines a networked analysis tool with airgapped generation tools, a tension examined further in Section 7.2.

5.1 OA Response Builder

Responding to an Office Action requires assembling a document with a specific, rigid structure: a caption identifying the application and the action being responded to, a claims-amendment section following precise USPTO formatting conventions, a remarks section, and a conclusion. Much of this structure is boilerplate driven by a handful of variables — the application number, the examiner, the type of action, the docket numbers — but assembling it correctly by hand is repetitive and error-prone.

The OA Response Builder generates this skeleton from a compact set of inputs (Figure 8).

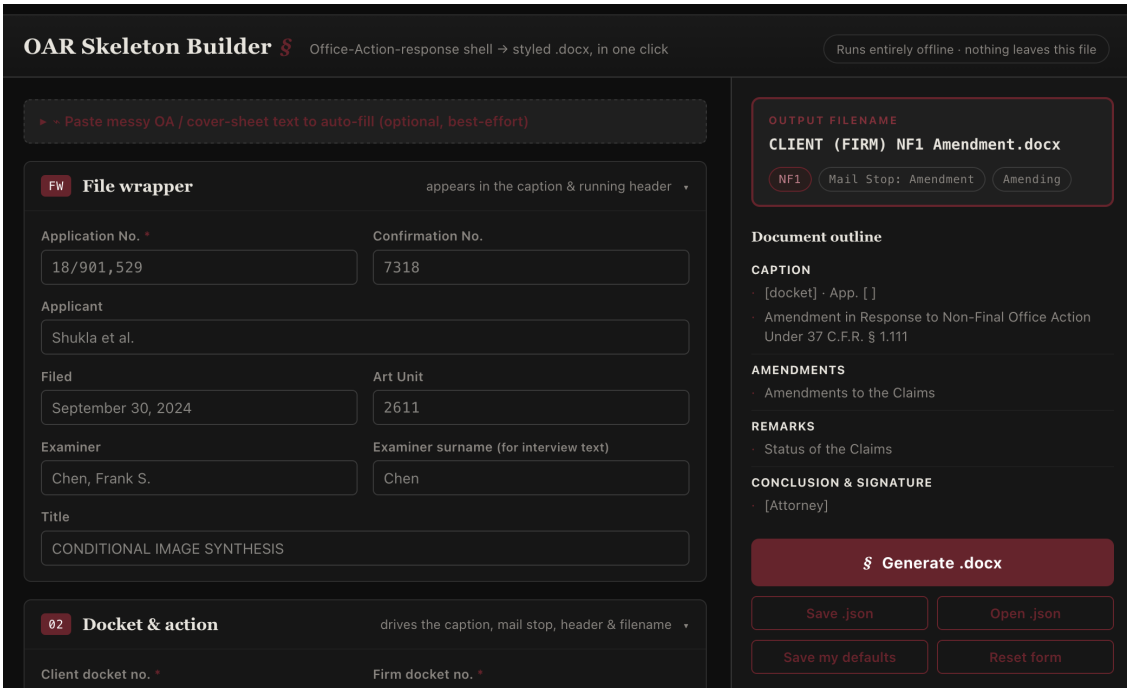


Figure 8: The OA Response Builder. From a compact set of file-wrapper and action inputs, the tool assembles a correctly structured, firm-styled Office Action response skeleton, with a live document outline and a derived output filename. It runs entirely offline.

The practitioner supplies the file-wrapper details and the action type, and the tool produces a correctly structured, firm-styled .docx in one click, with the caption, mail-stop, running header, and output filename all derived from those inputs. A live document outline previews the structure before generation. The tool embeds the firm's template directly, so the output matches house formatting without any external dependency. Because the generation is deterministic and entirely local, the tool functions as a fast, reliable starting point that a practitioner then fills with substantive argument. This tool removing the mechanical overhead of setting up the document while leaving all substantive content to the attorney, and is faster and less prone to errors than editing a previous document, as was the norm for years in my firm.

5.2 First-Print

Where the OA Response Builder addresses prosecution, First-Print addresses preparation: producing the first-draft skeleton of a new application specification. It, too, is an entirely offline, browser-only tool that embeds the firm's template and emits a styled .docx (and a Visio-compatible

.vsdx for drawings). First-Print organizes an application's raw materials, including matter metadata, claims, figures, element names, and reusable boilerplate, into the structured shell of a specification. One representative capability is its handling of claims: a practitioner pastes the claim set verbatim, and the tool parses it into structured claims, automatically detecting each independent claim and classifying it by type (method, system, apparatus, or computer-readable medium), while never rewriting the claim text itself (Figure 9).

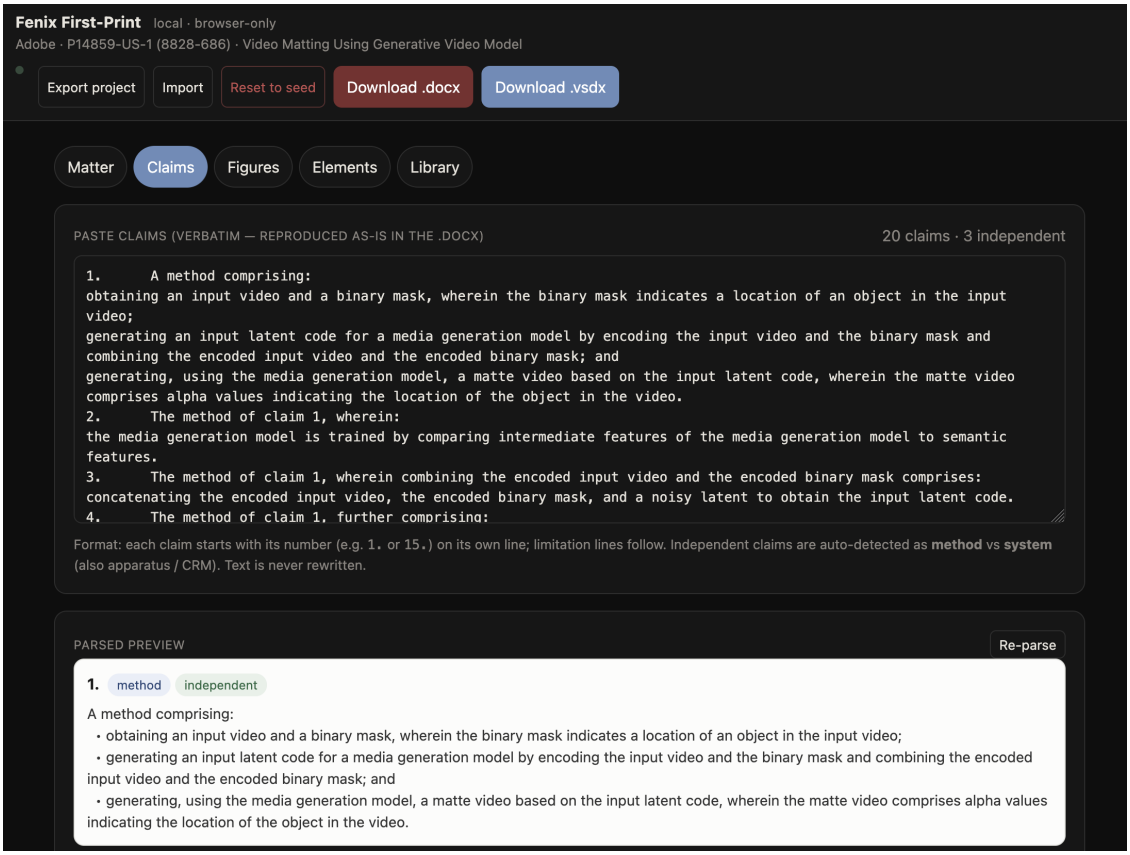


Figure 9: First-Print's claim handling. A pasted claim set is parsed into structured claims—independent claims auto-detected and classified by type—without rewriting the claim text. Here twenty claims are recognized, three of them independent.

'Fenix' is a nod to one of my supervising attorneys, as his company Fenix made a similar tool. This approximation was made with his blessing. It reports summary statistics — in the illustrated case, twenty claims of which three are independent — and renders a parsed preview that confirms the structure the practitioner will get in the generated document.

First-Print also incorporates a reusable boilerplate library, into which common architectural descriptions — for machine-learning inventions, components such as Transformer, U-Net, and diffusion-transformer architectures — can be maintained and inserted. This directly realizes a feature anticipated in the original project proposal: a one-click library of frequently-cited technical descriptions that removes redundant drafting effort. Notably, the library system is the component that most directly foreshadows the agentic direction of Section 7.2, since it establishes a structured, retrievable store of the domain descriptions a drafting agent would need.

5.3 Integration and hosting

The three tools were developed separately and retain their independence — each is a standalone file that functions on its own — but they are unified for the user through a shared navigation bar rendered identically across all pages, and a landing page that introduces each tool (Figure 7). The shared navigation is injected by a single small script that each tool includes with one added line, leaving every tool's internal logic untouched. The visual language — a dark serif theme — is applied consistently across the suite, including a re-theming of the two offline tools' interface chrome to match, while deliberately preserving light, readable "paper" surfaces for the document-preview areas where a practitioner reads generated text.

The result is that a set of independently-built tools presents as one coherent application, hosted at a single location, without sacrificing the standalone, offline character that gives two of the three their particular value. This integration is intentionally shallow at the code level — the tools do not share state or logic — which keeps each tool simple and independently maintainable, but it also marks the current boundary of the system: the tools sit side by side, and a human carries information between them. Removing that boundary is the subject of the final section.

6 Evaluation

The review tool was evaluated by practicing patent attorneys during a trial period at my firm, Chau IP. The evaluation had two parts: a quantitative helpfulness rating and open-ended qualitative feedback. The quantitative sample is small (six responses) and the results should be read as a directional signal rather than a rigorous measurement, but the qualitative feedback proved the more valuable of the two.

6.1 Method

Attorneys who used the tool during the trial were asked to rate its overall helpfulness on a four-point scale: 0 (not helpful), 1 (helpful), 2 (very helpful), and 3 (must-have before filing). They were separately invited to give open-ended feedback on what worked, what did not, and what they would want added. Six attorneys responded to the rating; their qualitative comments were collected alongside.

6.2 Quantitative result

The six responses were 2, 2, 3, 2, 1, and 2, for a mean of 2.0. This is squarely at "very helpful," with one respondent rating it "must-have before filing" and none rating it below "helpful." With a sample this small and drawn from a single firm, the result cannot support strong claims; it is best understood as an encouraging directional signal, and as a baseline against which future iterations can be compared. No respondent found the tool unhelpful, which given that the alternative to using it is the status quo of manual review, is a meaningful floor even if the ceiling is unmeasured.

6.3 Qualitative feedback

The open-ended responses were more useful than the ratings, both because they were specific and because they pointed directly at the tool's next priorities. The most consistent theme was not a complaint about the analysis quality but a request about currency: attorneys wanted assurance that the tool's understanding of the law stays aligned with an evolving landscape — new case law, updated USPTO guidance, and firm- or client-specific drafting preferences. In other words,

the feature most requested was a mechanism for keeping the system's legal knowledge current and locally customizable, rather than any change to the core review behavior. This feedback shaped the roadmap described in Section 7, and it is arguably the single most actionable output of the evaluation.

6.4 Limitations

Several limitations bound these results. The sample is small (n = 6) and drawn from a single firm with a shared house style, so the ratings may not generalize. I am both the tool's developer and a member of the evaluating firm, which introduces potential bias. The evaluation measured perceived helpfulness rather than any objective outcome, such as a reduction in Office Action rejections or in prosecution cost; establishing that kind of outcome would require a controlled, longitudinal study well beyond the scope of this project.

A further limitation is intrinsic to the underlying technology and warrants explicit mention. The analysis is produced by a large language model, which can occasionally generate plausible but incorrect assertions — including, notably, misattributed or non-existent case citations. When the tool cites a specific authority (as in the SAP America v. InvestPic reference shown earlier), that citation is a starting point for the practitioner's own verification, not a substitute for it. The system is designed and intended as a drafting aid that surfaces issues for a licensed practitioner to evaluate, and every substantive output — legal citations especially — must be independently verified before it is relied upon. This is a reason the tool's suggestions are deliberately sparse and its role deliberately advisory, as described in Section 4.6.

7 Future Work

The system as built is a working tool that practitioners found helpful, but there is plenty of room for iteration. This section describes two directions: near-term features driven directly by the evaluation feedback, and a longer-term architectural shift toward agent-assisted drafting.

7.1 Feedback-driven features

The most consistent request from the evaluation was for a mechanism to keep the tool's legal knowledge current and locally customizable — aligned with new case law, evolving USPTO guidance, and firm- and client-specific drafting preferences. This is a well-formed problem with a natural implementation.

Rather than baking legal knowledge into the application, the system can draw on an external, firm-maintained instruction document — a structured guidance file that the model consults as part of its analysis. Recent developments in guiding language-model behavior through supplied instruction files make this practical: a firm could maintain a single authoritative document capturing its house positions on recurring issues (how it prefers to handle § 101 eligibility framing, which claim constructions it favors, client-specific constraints, and current examination guidance), and the tool would incorporate that document at analysis time. The crucial property is that this file is maintained by the firm, not the developer: it can be updated the day a significant Federal Circuit decision issues or a client changes its preferences, without any change to the application itself. This directly answers the currency concern — the tool's legal understanding becomes a living document under the firm's control rather than a fixed property of the code. A second, complementary direction is integration with the firm's existing systems. The firm maintains a master docketing platform, Fenix, which already exposes a connector interface to the same model provider the review tool uses.

Two integrations follow naturally. First, the tool could synchronize the documents it analyzes with the docketing system, so that a review is automatically associated with the correct matter rather than handled as a loose file. Second, and more powerfully, the firm's authoritative guidance — the currency mechanism above — could be sourced from or checked against the docketing system, so that the single source of truth for "the firm's current position" lives where the firm already manages its matters. This would fold the review tool into the firm's established workflow rather than standing beside it.

Two smaller features round out the near-term roadmap. A bring-your-own-key capability would let each attorney supply their own model-provider credentials, so that internal adoption does not route all usage through a single account — a straightforward change given the provider abstraction already in place. And the patent-specific grammar and style checking anticipated in the original proposal — flagging "patent profanity" such as the words invention and preferred embodiment, and vague antecedents that weaken a claim — remains a valuable, largely rule-based addition that would complement the substantive statutory analysis with stylistic review.

7.2 Agentic drafting

The deeper opportunity, and the one that most clearly extends the present work, is a shift from analysis to assisted drafting. The current suite is a pipeline whose stages a human connects by hand: the review tool analyzes, First-Print assembles a specification skeleton, the OA Response Builder assembles a response skeleton, and a practitioner carries information between them. An agent could connect those stages directly.

The concrete proposal is a drafting agent that produces a substantially complete first-draft application from an inventor's raw disclosure. The agent would be handed a structured description of the matter — a JSON representation of the application's state, the formalized drawings, the underlying disclosure material (an inventor's technical write-up, a paper, or notes), and a rough mapping of which disclosure content corresponds to which figure — and would work toward a specification draft that is 80–90% of the way to complete: figures described, components named consistently, the detailed description drafted against the drawings, and the boilerplate architectural descriptions inserted where they apply. The remaining 10–20% — the genuinely inventive framing and the strategic claim decisions — would remain with the attorney, which is both where the value is and where a human must stay.

What makes this proposal tractable rather than speculative is that most of the machinery already exists in the present system, and would be reused as the agent's tools. The document parser already converts between .docx applications and structured representations. The validator already checks structural completeness. First-Print's boilerplate library already provides the retrievable store of architectural descriptions an agent would insert. Reframed as callable functions available to a planning agent — parse this document, validate this structure, retrieve this architectural description, generate this skeleton — these deterministic components become the reliable tools that an agent orchestrates, with the language model supplying planning and drafting judgment rather than being asked to do everything in a single unreliable pass. This is the standard and effective shape of an agentic system: a model that plans and iterates, calling dependable tools to act on real state, and critiquing its own output against structural checks before producing a result.

One genuine architectural tension must be acknowledged. Two of the three existing tools are deliberately offline — their value depends on sensitive draft material never leaving the file. A drafting agent, by contrast, requires model access, which is inherently networked. Reconciling these is a real design question rather than a detail: the resolution might confine the networked, generative work to a clearly-scoped component while the offline tools remain airgapped, or might depend on

the availability of locally-run models capable enough for the drafting task. This tension is not a reason to avoid the agentic direction, but it is the first problem that direction must solve, and naming it honestly is part of designing it well. A prototype of this agent is intended to be developed on a separate branch of the project, kept off the critical path of the present deliverable. Its purpose is to demonstrate the handoff design and the reuse of existing tools as agent tools, and to surface, in practice, the offline/online reconciliation described above.

8 Conclusion

This project began from a concrete, daily frustration in patent practice: that a valuable pre-filing review is expensive and time-consuming, and that the tools available to assist it either address only shallow formal properties or cannot reliably ground their feedback in the document at all. The system described here addresses that gap with a pre-filing review tool that triages a complete application across the four principal statutory grounds for rejection and — critically — ties every observation to a specific, verifiable paragraph. The central technical contribution is the recovery of that grounding. Patent practitioners work in terms of paragraph numbers that Microsoft Word generates rather than stores, and which conventional extraction tools and even modern language models discard or hallucinate. By parsing the document's structure directly, the system reconstructs those numbers reliably — including for the restructured as-filed documents that dominate prosecution work — and thereby makes location-specific, verifiable feedback possible where it previously was not. Around that core, a two-phase analysis, a deliberately advisory suggestion design, and an integrated suite of preparation tools form a coherent system that practicing attorneys rated as very helpful.

Equally important is what the project clarified about direction. The evaluation showed that practitioners' foremost concern is not analysis quality but currency — keeping the tool's legal knowledge aligned with an evolving field — which points to a firm-maintained, living guidance mechanism as the highest-value next step. And the structure of the existing suite points naturally toward an agentic future in which the deterministic tools already built become the reliable instruments of a drafting agent, moving the system from identifying problems toward helping solve them. The work sits, deliberately, at the intersection of software engineering and patent practice; its most promising path forward is to deepen that intersection rather than to broaden it.