

# LoanShark Sentinel – Predatory Loan Scanner

## MVP: Product Requirements Document

### Executive Summary

**LoanShark Sentinel** is a real-time loan agreement scanner designed to expose predatory lending practices in under 60 seconds. Predatory loan contracts often hide exploitative terms that burden borrowers with exorbitant costs, unfair repayment conditions, aggressive collection powers, waivers of consumer rights, or deceptive fine print <sup>1</sup>. This MVP addresses that problem by leveraging machine learning and rule-based pattern recognition to **flag high-risk clauses and compute a “Predatory Risk Score”** for any loan document. The goal is to empower everyday borrowers (who lack legal or financial expertise) with a quick, clear assessment before they sign a potentially harmful loan.

**Why now?** In the current economic climate, many people resort to high-cost payday or installment loans for urgent needs, often without the ability to fully decipher the dense legal jargon. Regulators and consumer advocates are cracking down on predatory lending, but individuals still need a tool to protect themselves in the moment. Recent advances in **OCR, lightweight ML models, and cloud-free computing** enable LoanShark Sentinel to operate quickly and privately (no internet or heavy AI needed), making it timely and feasible as an 8-day hackathon project. In summary, LoanShark Sentinel tackles a pressing consumer finance issue with an innovative blend of **fast OCR, rule-based legal analysis, and financial calculations**, delivering an actionable risk score and explanations that can save borrowers from debt traps.

### Problem Background & User Persona

**The Predatory Lending Problem:** Predatory lending (e.g. payday loans, car title loans) routinely harms vulnerable borrowers. These loans carry sky-high interest rates (often 200–500% APR) and sneaky contract terms that lock borrowers into cycles of debt. A borrower might be hit with hidden fees, impossible repayment deadlines (like full repayment in two weeks), or clauses that let the lender withdraw money directly or sue without trial. The result is that financially stressed individuals – often with low financial literacy – get caught in a **debt trap**, refinancing or rolling over loans repeatedly because they can never catch up. The **real-world harm** includes drained bank accounts, repossessed cars, wage garnishments, and loss of legal recourse, all from fine-print clauses the borrower likely didn’t understand.

**Target User Persona:** *Meet Maria*, a 32-year-old working mother in need of a quick \$500 loan to cover an emergency. She finds a small lender willing to give her a short-term loan. The contract paperwork is long, full of legal terms, and **written at a college reading level** – far above Maria’s comfort (she reads at about a 9th-grade level). Maria is in a hurry (rent is due tomorrow), **under stress**, and doesn’t have a lawyer. Our target user is someone like Maria: a non-expert borrower with **urgent financial needs and limited literacy in legal/financial jargon**. They may suspect a loan offer could be unfair but lack the tools to verify it. In Maria’s context – sitting in a lender’s office or viewing an online loan agreement – she has just a minute or two to decide. **LoanShark Sentinel** is built for her: it lets her snap a photo of the contract or paste the text, and within a minute get a plain-language report of any predatory terms and an overall risk score. This immediacy and simplicity are critical, as our user often **must decide on**

**the spot** whether to take the loan or walk away. The persona's needs drive our focus on **clear explanations** (not legalese), **speed**, and an **offline-capable, private analysis** (since she may be using her phone in a location without reliable internet).

## MVP Functional Scope

For the 8-day hackathon MVP, we will implement a focused set of features that cover the end-to-end flow of scanning a loan document and producing a risk assessment. The emphasis is on **backend logic and ML integration**, with a minimal interface just to demonstrate the functionality (UI/UX polish is out of scope). Key functional components and user flows include:

- **Document Input (Text or Image):** The user can input a loan agreement either by uploading a document image/PDF or by copying & pasting text. We will support a basic web interface (or CLI) where the user provides the loan contract. If an image or scanned PDF is provided, the system will invoke a **Tesseract OCR** pipeline to extract text from the scan. *(OCR is optional in MVP, but we plan to include it for real-world realism. If OCR fails or time is short, the user can fall back to manual text input.)*
- **Clause Detection Engine:** Once text is obtained, LoanShark Sentinel parses it for any **tell-tale predatory clauses** using a library of regex patterns and keyword checks. This is the rule-based core: the backend will scan the text and flag sections that match our known high-risk patterns (details in Clause Library below). For each detection, the system will record the clause category (e.g. "Collection/Enforcement" or "Legal Waiver"), the exact snippet of text, and an internal flag (e.g. `has_arbitration_clause = True`). This step also includes basic **financial data extraction** – for example, if an interest rate or fee is mentioned, we capture that numeric value for calculations. The clause detection runs in well under the target time (regex matches on text are very fast), ensuring the analysis stays within our ~60 second budget even on documents a few pages long.
- **Predatory Risk Scoring:** The backend then computes an overall **Predatory Risk Score (0-100)** for the loan. This involves a **hybrid approach**:
  - A **rule-based scoring** function assigns points for each risky element found. (For instance, an APR above 300% might add +30 points, an omitted APR disclosure +10 points, and each legal rights waiver +5 points <sup>2</sup> <sup>3</sup>.) We will implement a scoring rubric that sums these points in a transparent way.
  - In parallel, the system can feed the extracted features into a lightweight **ML model** (e.g. logistic regression or XGBoost) which predicts a risk level based on patterns it learned from example loans. This ML output can be combined with the rule score (for MVP, we might average the two or use the ML as a validation layer). The result is a single risk score and category label (e.g. **Safe**, **Caution**, **High Risk**, or **Predatory**) that reflects the document's overall riskiness.
  - The scoring step is designed to be **fast and explainable**. The ML model is small (few kilobytes if in ONNX format) and runs inference in milliseconds, and the rule scoring is just arithmetic on the detected flags. We will ensure this logic is executed within the request-response cycle of the FastAPI backend, keeping total processing time well under one minute.
- **Results & Explanation Output:** Finally, the system returns a result to the user containing **(a) the Predatory Risk Score (with a risk category)**, and **(b) a breakdown of flagged clauses/terms with plain-language explanations**. For the MVP demo, this could be a JSON output from the API or a simple web page listing the findings. For example, the output might say: *"Predatory Risk*

*Score: 85 (High Risk). Key flags: Mandatory arbitration clause (you waive your right to sue), 450% APR (extremely high cost), automatic bank withdrawals (lender can drain your account), missing clear APR disclosure.*" Each flagged clause will be tied to the actual text snippet (e.g. by highlighting it in the output or listing quotes) so the user can see where it came from in their document. The explanations will be concise and in non-technical language, suitable for the average borrower's understanding. This closes the loop of the user flow – from inputting a contract to receiving an actionable analysis – all in about a minute.

The above scope covers the core functionality needed to demonstrate our value proposition at the hackathon. We intentionally limit the MVP to the essential backend logic: text parsing, rule-based checks, ML inference, and output generation. Features like user accounts, data storage, or a polished UI are **out of scope** for the MVP. We will present the functionality likely via a simple web form or notebook interface during the demo. The focus is on proving that **"real-time predatory loan scanning and scoring"** is achievable and effective.

## Clause Library & Risk Logic Summary

A cornerstone of LoanShark Sentinel is its **Clause/Pattern Library** – a curated set of patterns that represent the most common predatory lending clauses. We have organized these into **six key categories** of risk. For each category, the system uses regex patterns and keyword rules to detect problematic language, and each detection contributes to the risk score. Below is an overview of the 6 clause categories, with their predatory significance and examples of how we detect them:

- 1. Excessive Cost (Interest & Fees)** – Predatory lenders often impose **exorbitant interest rates and fees** that far exceed normal loans. The scanner flags any indication of sky-high costs or cleverly hidden charges. **Detection patterns:** Look for explicit triple-digit APR percentages or fee descriptions that imply such rates. *Example:* The phrase "Annual Percentage Rate: 450%" would be caught as an extreme APR <sup>4</sup>. Even if the contract avoids the term APR, patterns like "\$15 fee per \$100 loaned bi-weekly" would trigger a flag, since our logic can compute that equates to an ~390% APR (a predatory range). We also flag **excessive fees** layered on the loan (e.g. origination or service fees >5% of the principal <sup>5</sup>). Each costly element (high interest, large fees, daily compounding interest, etc.) raises the risk score. For instance, an APR above ~300% might add a big chunk of points in our rubric, given how **extreme cost** is a primary indicator of predation.
- 2. Unfair Repayment Terms** – This category covers contract terms that make repayment unreasonably difficult, virtually ensuring the borrower will default or have to refinance. **Detection patterns:** Flag loans with **very short repayment periods or lump-sum payments** that are unrealistic for the borrower. *Example:* "Payment due in full in 14 days" would be identified as a red flag, since a two-week term for full repayment is often impossible for cash-strapped borrowers <sup>6</sup>. We also catch **balloon payments** (a final massive payment after a series of small payments <sup>7</sup>) and **interest-only payment periods** (where the contract states that payments only cover interest, not principal <sup>8</sup>). Another unfair term is a **prepayment penalty** – a fee for paying off the loan early, which traps the borrower from escaping the debt; keywords like "prepayment penalty/fee" are detected and flagged <sup>9</sup>. These repayment structures are highlighted to the user with notes like "This loan demands full repayment very quickly, which is a common debt trap". In scoring, the presence of any such clause adds risk points (e.g. a balloon payment or prepayment penalty might each add several points) because they indicate the lender isn't setting the borrower up to succeed.

3. **Debt Cycle Triggers (Refinance & Rollovers)** – Predatory lenders often design loans to **keep borrowers in a cycle of debt**. This category overlaps with repayment terms but focuses on clauses that explicitly allow extending or refinancing the loan in ways that pile on more fees. **Detection patterns:** Keywords like “rollover”, “renewal fee”, or “extend for an additional term” are strong signals <sup>10</sup>. *Example:* A clause stating “Borrower may renew this loan up to 6 times for a fee” would be flagged immediately. Our system notes both the existence of a rollover option and the number of times allowed <sup>11</sup> – multiple renewals are a hallmark of predatory loans trapping borrowers in debt. **Automatic rollovers** (where the contract says the loan will automatically renew if not paid on time) are also caught by phrases like “will be automatically renewed” <sup>12</sup>. Each such feature contributes to the risk score because it indicates the lender expects (or even encourages) the borrower to **refinance repeatedly** and pay far more in the long run. We may, time permitting, even calculate the theoretical total cost if a loan is rolled over the maximum times and include that in the explanation.
4. **Aggressive Collection & Enforcement** – Predatory contracts often give lenders **extraordinary powers to collect payment or collateral**, bypassing normal consumer protections. In this category, the scanner flags clauses that let the lender grab the borrower’s money or property or harass the borrower in ways legitimate lenders typically wouldn’t. **Detection patterns:** We look for terms related to automatic bank account debits, wage assignments, security interests, and extreme default penalties. *Examples:* A phrase like “authorize Lender to debit your bank account” is caught and flagged <sup>13</sup>, with an explanation that the lender can pull funds directly from the borrower’s account (risking overdrafts and loss of control). **Wage assignment** clauses (e.g. “irrevocable wage assignment” <sup>14</sup>) are identified, meaning the lender can take a portion of the borrower’s paycheck without a court order – essentially a quasi-garnishment that we warn the user about. If the contract mentions **collateral or security interest** (e.g. a lien on the borrower’s car <sup>15</sup>), we flag that the borrower’s personal property is at risk of being seized. Clauses like “confession of judgment” (where the borrower pre-authorizes the lender to win a court judgment if they default) are detected by keywords like “confess judgment” <sup>16</sup> – our tool would highlight this and explain it means the borrower gives up the right to defend themselves in court. We also catch any permission the borrower gives for the lender to **contact third parties** (employers, relatives) for collection <sup>17</sup> or any clause making the borrower pay for collection costs and attorney fees <sup>18</sup>. Each of these aggressive collection clauses raises the risk score (e.g. auto-ACH debit +5 points, wage assignment +5, confession of judgment +5, etc.), as they indicate the lender will use hardball tactics that can seriously harm the borrower’s finances or privacy if they fall behind.
5. **Legal Trap Clauses (Waivers of Rights)** – Predatory loan agreements often include fine-print **legal waivers that strip borrowers of important rights**, protecting the lender from lawsuits or accountability. Our scanner will flag any clause where the borrower **waives a legal right or agrees to a biased legal process**. **Detection patterns:** Keywords around arbitration, class-action waivers, jury trial waivers, and choice-of-law clauses are key. *Examples:* “All disputes will be resolved by binding arbitration” would be highlighted as a **mandatory arbitration** clause <sup>19</sup>. The app would explain that this means the borrower cannot sue the lender in court, which heavily favors the lender. A phrase like “borrower waives the right to participate in any class action” indicates a **class action waiver** <sup>20</sup> – we flag that because it prevents borrowers from banding together to challenge the lender. We also look for any **jury trial waiver** (“waiver of jury trial” in the text) and **choice of law/venue** clauses that force disputes into a distant or lender-friendly jurisdiction <sup>21</sup> (for instance, a loan for a borrower in New York stating it’s governed by Utah law – known for weaker usury laws – would be flagged and explained). Additionally, clauses that **limit the time to sue or the damages recoverable** (like “must bring any claim within 1 year” or “no consequential damages”) are caught as they tilt legal recourse in the lender’s favor <sup>22</sup>. Each

of these legal traps adds to the risk score (we assign moderate points, e.g. +5 for each waiver like arbitration or class-action, since these significantly reduce borrower protections <sup>3</sup>). The user sees a note like *“Mandatory arbitration clause – you’re giving up your right to take the lender to court.”* The presence of any of these waivers is a strong indicator of a predatory contract ethos.

6. **Hidden or Deceptive Disclosures** – Predatory contracts often **obfuscate key information** or bury it so the borrower can’t easily find or understand it. This category flags issues of transparency and clarity. **Detection patterns:** We check if required info is **missing or masked** and if the language is overly complex. *Examples:* If **no APR is stated at all** in a loan that clearly charges interest, that’s a huge red flag – our parser searches for “APR” or “Annual Percentage Rate” and if not found, we flag a **missing APR disclosure** <sup>23</sup>. (Omitting the APR is often illegal and suggests the lender is hiding the true cost; our output would explicitly warn “APR not disclosed – the true cost of the loan is hidden.”) We also catch **vague fee descriptions** where fees are described in misleading ways – e.g. calling interest a “service fee” or splitting interest into multiple innocuous-sounding fees (“maintenance fee”, “credit access fee”) <sup>24</sup>. Regex patterns will look for the word “fee” preceded by terms like “service, maintenance, processing” etc., to flag such **opaque fees**. Similarly, any **undefined terms or open-ended language** (“additional fees may apply”, “interest rate may change at lender’s discretion” <sup>25</sup>) are flagged as ambiguous traps – the tool will note that the contract leaves room for surprise charges or changes. We’ll also flag references to **external documents** not provided in the contract (e.g. “See attachment A for fee schedule” <sup>26</sup>), since that means the borrower doesn’t have all the info at hand. Finally, we consider the possibility of **excessive legal jargon or tiny fine print** – while harder to detect automatically, our OCR module can give a confidence score or detect if large sections of text were unreadable (lots of OCR errors may imply the scan was of fine print) <sup>27</sup>. If we detect that the document text quality is poor or convoluted, we may add a cautionary note like “Important terms might be hidden in hard-to-read fine print.” Each transparency issue (especially a missing disclosure) adds significantly to the risk score, because lack of transparency is itself a predatory tactic (e.g. not stating the APR might add +10 points in our rubric <sup>28</sup>).

**Risk Scoring Logic:** All the above detections feed into a unified scoring system that produces the Predatory Risk Score (0–100). The scoring is **explainable and rule-based** so that we can justify it to users and judges. We allocate a maximum number of points to major risk areas: **Cost, Clauses, and Transparency**. For example, cost-related factors (APR, fees) might contribute up to ~30–40 points: an APR well above 300% could add +30 points by itself (extremely high risk range) <sup>29</sup>, while a moderate but still high APR (say 80%) might add +10. Multiple fees beyond the interest (origination fee, monthly fee, etc.) each add a few points depending on size <sup>30</sup>. Predatory clauses (categories 3, 4, 5 above) each add a smaller chunk of points; we weight especially egregious ones like *confession of judgment* or *arbitration* slightly higher (around 5 points each) <sup>3</sup>, so a contract loaded with dangerous clauses racks up points. Transparency issues also add points – e.g. completely missing an APR disclosure is given a heavy penalty (+10) because that’s a strong sign of deception <sup>28</sup>. Lesser issues like one instance of vague wording might be +2 or +5. We will fine-tune these thresholds during development, but the principle is that points *accumulate* for every risk factor. The sum of points is then mapped to a qualitative **risk tier**: for instance, 0–20 = “Safe/Low Risk”, 21–50 = “Caution”, 51–80 = “High Risk”, 81+ = “Predatory”. The output will clearly show the numeric score and the tier name. By summing contributions from identifiable factors, our score is naturally interpretable: it’s easy to say “You got 75 points, which is High Risk, largely because of the 450% APR (+30), two rollover allowances (+10), an arbitration clause (+5), and missing APR info (+10), etc.” This transparency is important for user trust and for judges to see that our scoring isn’t a black box.

## ML Plan & Integration

While the rule-based engine covers known patterns, we plan to integrate a **machine learning component** to enhance detection and scoring. This **hybrid ML approach** will help catch subtle signals and provide a statistically grounded risk prediction to complement the rules. Our strategy is to use **lightweight, interpretable models** that can be trained quickly (within Colab) and run efficiently in the FastAPI backend without external dependencies (no large language models or cloud calls required).

**ML Architecture:** We will implement a simple **classification model** (likely Logistic Regression or XGBoost) that takes in a set of features extracted from the loan text and outputs a risk assessment (either a probability of being predatory or a risk score). Logistic Regression is appealing because it's fast and its coefficients provide insight (which features contribute most) <sup>31</sup> <sup>32</sup>. XGBoost or a small Random Forest is an alternative if we find some non-linear interactions (e.g. a combination of moderate APR + multiple rollovers might together signal high risk even if each alone is borderline <sup>33</sup>). In either case, the model will be kept **small and fast** – we expect on the order of tens of features, so even a simple model will run inference in milliseconds. We'll train the model in Google Colab (leveraging scikit-learn or XGBoost libraries) and export it as a **pickled model or ONNX** file for easy loading into our FastAPI server <sup>31</sup>. This means the ML model can run locally with no internet and minimal memory, aligning with our hackathon constraints.

**Feature Engineering:** A critical part of the ML integration is defining the input features for the model. We will **transform each loan document into a numeric feature vector** that captures the key aspects of the text <sup>34</sup>. Based on our clause library and domain knowledge, likely features include:

- **Cost features:** The numeric APR (if found), or an estimated APR computed from fee terms; total dollar fees and total fees as a percentage of the loan amount; the presence of certain fee patterns (e.g. a flag if we saw a “fee per \$100” phrase). For example, if the contract says “Finance Charge: \$60 on \$200 loan for 14 days”, we'd compute ~780% APR and feed that number in as a feature.
- **Clause flags (binary or count):** For each predatory clause category (and sub-clause) that our rule engine checks, we create a feature. For instance, `has_arbitration_clause`, `has_class_action_waiver`, `has_auto_rollover`, `has_confession_of_judgment`, `has_wage_assignment`, etc., each set to 1 if the clause was detected <sup>35</sup>. We can also include counts, like how many times did we find the word “fee” or “penalty” – multiple occurrences might indicate a fee-heavy document <sup>36</sup>.
- **Transparency flags:** Features indicating missing info, such as `no_APR_mentioned` (1 if our regex found no APR) or `no_clear_repayment_schedule` if applicable <sup>37</sup>.
- **Document complexity metrics:** We can include the length of the document (word count or character count) and a readability score (like Flesch-Kincaid grade level or average sentence length) <sup>38</sup>. Hypothesis: predatory contracts might be extra verbose or use complicated language to confuse borrowers, which a reading level feature could capture.
- **Derived ratios:** If data allows, compute things like the loan term length (in days) vs. the amount of fees, or how much the borrower has to pay per week relative to principal. For example, a “cost per day” of loan or other ratios that might differentiate a normal loan from a predatory one (though these might overlap with APR) <sup>39</sup>.

All these features will be assembled into a vector for each document. For instance, after parsing a given contract, we might produce something like: `APR=400, total_fees_percent=12%, has_arbitration=1, has_wage_assignment=0, fee_word_count=3, no_APR_flag=0, wordcount=1500, readability_grade=8th` <sup>40</sup>. We expect truly predatory loans to have tell-tale combinations (e.g. extremely high APR, multiple predatory clauses, lots of fee words, etc.), whereas a standard personal loan from a bank would have low APR, maybe none of those nasty clauses, and clearer disclosures.

**Training Strategy:** We will need a labeled dataset of loan agreements to train the model. Given the time constraint, we'll start with a **small hand-collected dataset**: - We plan to gather ~40-50 examples of loan agreements or snippets. Sources might include public domain sample contracts, known payday loan contracts available online, and perhaps consumer financial protection databases. For instance, the CFPB has a repository of complaints – we could use payday loan complaint narratives or known bad contract examples as “predatory” instances, and use standard bank loan terms or sanitized contracts as “non-predatory” instances. We may also generate some synthetic examples by tweaking terms in a safe loan contract to make it predatory. - We will likely label them in a binary way (predatory vs not predatory) or possibly a 3-class (low, medium, high risk) for training. Even with a small dataset, a logistic model should pick up on obvious signals (e.g. triple-digit APR + arbitration clause = predatory). Our rule-based features essentially embed expert knowledge, so even few samples can validate the weighting. - Using Colab, we'll train the model on this dataset, tune it lightly (not much time for extensive hyperparameter tuning, so we'll choose defaults or simple grid search if needed), and evaluate it. If performance is decent (it correctly flags predatory ones and not the safe ones), we'll proceed to integrate it. If it's too inconsistent, we may simplify to binary flags or rely more on the rule engine for final scoring in MVP.

**Integration with Backend:** In the FastAPI backend, after the rule-based analysis runs, we will have the necessary inputs for the ML model (the feature vector). We'll load the pre-trained model at startup (e.g. using joblib to load a .pkl or ONNX runtime to load a model) so that inference is just a function call. During a document analysis request, the backend will: 1. Parse the text and gather features (as described). 2. Pass the feature vector to the ML model to get a prediction (e.g. a probability that the loan is predatory, or a raw risk score). 3. Combine this with the rule-based score. There are a few ways to combine: - **Hybrid score combination:** We might translate the ML prediction into a 0–100 scale and then average it with the rule score for a final number. - Alternatively, we could use the ML prediction to adjust the rule score: for example, if the ML model predicts very high risk but our rules scored somewhat lower (perhaps because the loan had an unusual combination of red flags that the rules didn't fully capture), we can bump up the final score. Conversely, if ML predicts low risk but rules scored high, it might prompt us to double-check if any false positives occurred. For MVP simplicity, a weighted average or max could be used (ensuring we don't ever downplay a serious risk). 4. The backend then generates the output JSON, including the final score and the list of flagged reasons (the rule-based explanations). The ML model's influence on the score is behind the scenes – we will **surface explanations based only on actual clauses and data from the document**, not on opaque ML features, to maintain user trust. The ML is there to improve accuracy and catch edge cases, not to replace explanation.

Overall, the ML component is an **assistive layer**. By combining deterministic rules with a trained model, we get the best of both: **precision** on known bad patterns and **recall** on subtle patterns. This approach is robust and suitable for the hackathon: the models are small and fast, and the logic is transparent. We avoid the pitfalls of large black-box models and ensure the core functionality (rule engine) works independently if the ML were to misfire. This means even if the ML integration isn't perfect by demo time, our product will still function with the rule-based scoring alone (and we can show the ML as an enhancement). But with successful integration, we'll be able to proudly say we built a **hybrid rule/ML system** that is more resilient than either approach alone <sup>41</sup>.

## Demo Stability Tactics

Given the live demo constraints (must *always* produce a result quickly and reliably), we have built in several measures to ensure **stability, speed, and confidence** in LoanShark Sentinel's output. Our

strategy is to minimize points of failure and handle uncertainty gracefully, so the demo runs smoothly under various conditions. Key tactics include:

- **Offline-First Operation:** The entire pipeline (OCR, text processing, ML inference) runs locally without internet access. This eliminates reliance on external APIs or networks that could lag or fail during the demo. For example, using Tesseract for OCR and a local ML model means even with no Wi-Fi, the analysis works. This also means **user data never leaves the device**, a bonus for privacy and a point we can highlight in the demo (important for a fintech tool).
- **Performance Optimizations:** We are choosing algorithms and models with speed in mind. Regex pattern matching and straightforward string searches are extremely fast in Python for our document sizes. The ML model is lightweight (logistic regression or similar) with a small feature set, so inference is virtually instant. We will load the ML model into memory ahead of time (at server start) to avoid any loading delay on first request. If using FastAPI, we can leverage asynchronous calls – for instance, run OCR and text analysis concurrently if needed – though the sequential steps are likely fast enough. We will test with sample documents to ensure we consistently stay well under the 60-second mark (we aim for ~10-20 seconds for most inputs). If any step threatens to be slow (say a very large document), we will set timeouts or chunk processing to avoid hitting a wall. Our goal is that judges can literally watch us scan a document and get results almost immediately, underscoring the “**real-time**” claim.
- **Robust OCR and Parsing:** Scanned documents can vary in quality. To avoid OCR hiccups, we’ll use Tesseract’s configurable options (like setting appropriate DPI or language hints) to maximize accuracy. If the OCR text confidence is low or we detect garbled output (e.g. lots of unknown characters), the system will not crash; instead, it will output what it *could* read and flag that the scan quality was poor. For instance, we might include in the results something like: “⚠ The document scan was low quality; some text might not have been read correctly.” This serves as a **low-confidence flag** to inform the user that the analysis might be incomplete, rather than simply giving a wrong or partial result without warning. In a demo, this shows we’ve thought of edge cases and adds credibility.
- **Graceful Degradation & Fallbacks:** Every component has a fallback mode. If the ML model for some reason doesn’t load or throws an error, we will catch that and fall back to just rule-based scoring (and perhaps note “ML model unavailable, using rule engine only” internally, though in the user-facing output the difference is not visible). If OCR completely fails on an image (e.g. a very blurry photo), we’ll return a message prompting the user to provide text input instead, rather than the app hanging. The rule engine itself is designed to handle missing data: e.g., if no APR is found, it doesn’t break – it sets a flag (missing APR) and continues. By designing the pipeline in modular steps with error-handling at each boundary, we ensure one failure doesn’t cascade into a full system failure.
- **Consistent Scoring and Banded Results:** To avoid confusion and ensure the output is easy to interpret in a demo, we use **score bands** (Safe/Caution/High Risk/Predatory) as described. This means that whether a loan scores 82 or 90, the audience will simply see “Predatory (risk score 82)”, etc. The exact number is there for precision, but the label keeps the message high-level. This prevents any small variability (say if we tweak a weight last-minute) from affecting the clarity of the result. It also ensures the tool always provides a **clear recommendation** – e.g. “Caution: consider avoiding or renegotiating this loan” – rather than just raw numbers. In testing, we’ll calibrate these bands so that they make intuitive sense. For example, we might decide that anything over 80 is “Predatory” and in practice that would be loans with multiple severe issues. By the demo, we’ll have examples for each category to show consistency.

- **Confidence and Explainability in Output:** If some detections are borderline (perhaps the regex is unsure, or a value is just at a threshold), we prefer to err on the side of caution in the messaging. The app might phrase a finding as “Possible issue: ...” if it’s not absolutely certain, rather than failing silently. For the hackathon demo, we will prepare a couple of known test documents to run through, ensuring we can demonstrate both a relatively *safe* loan and a blatantly *predatory* loan and show the output differences. This testing will also help us refine any final stability issues (for example, if a particular regex misfires on certain phrasing, we can adjust it or at least be aware of it). By demo day, LoanShark Sentinel will have been run on a variety of sample contracts so we’re confident it produces coherent, accurate results every time.

In summary, our stability tactics revolve around **keeping the system simple, local, and fault-tolerant**. We’ve avoided external dependencies and heavy models, we’ve built in checks for input quality, and we handle exceptions gracefully. This gives us a high degree of confidence that we can live-demo the MVP and get consistent results, which is crucial in a hackathon setting. If anything unexpected does occur, our fallback mechanisms ensure the app still delivers a meaningful output to the user instead of failing. The judges will see a polished, reliable tool in action.

## Differentiation

LoanShark Sentinel’s MVP distinguishes itself from existing legal tech or fintech solutions through several key differentiators:

- **Predatory Lending Domain Focus:** Unlike general contract analysis tools or do-it-yourself legal apps (which try to cover everything from leases to NDAs), our product zeroes in on **payday and high-cost loan agreements**. This specialization means we catch nuances others miss – for example, a generic document scanner might not flag a 200% APR as “high” or recognize a rollover clause as dangerous, whereas our system is built with deep knowledge of predatory loan tactics. This domain focus fills a gap where existing tools are either too broad to spot these specific red flags or too narrow (e.g. simple loan calculators) to interpret legal terms <sup>42</sup>.
- **Hybrid Rule+ML Engine with Explainability:** Many AI contract review tools rely on either pure machine learning (black box models) or simple keyword checkers. We combine the strengths of both. Our **rule-based component** provides **transparent, explainable results** – each finding is backed by a known pattern and we explain it in plain English. Meanwhile, the **ML component** adds an intelligent second layer, catching subtle patterns and weighting the severity of issues. This hybrid approach means we deliver **accurate results without sacrificing interpretability** <sup>41</sup>. For example, if we flag an arbitration clause, we can point to the exact sentence and explain its impact; if the ML flags the document as high risk, it’s because of the features derived from concrete data (like “multiple fees and high APR”). Competing products often lack this balance: rule-only systems miss complex interactions, and ML-only systems can’t explain *why* they think a contract is bad. Our approach gives users and judges the “*why*” behind the score, every time.
- **Holistic Analysis (Legal + Financial):** LoanShark Sentinel bridges the gap between a legal analyzer and a financial calculator. We not only identify legal clauses, but also perform computations (APR, fee percentages) to put those clauses in context. For instance, if a contract says “\$50 fee on \$200 loan”, we calculate what that means for the effective interest rate and flag it as predatory pricing. Traditional fintech tools (like loan calculators) might compute payments but **won’t warn you** that a “mandatory arbitration” clause lurks in the terms. Conversely, a legal document tool might highlight “arbitration clause” but **won’t compute** that the loan costs 400%

APR. By combining both, our MVP provides a **one-stop analysis** – it's like having a financial advisor and a legal expert simultaneously review the loan. This comprehensive view is unique and crucial for fully assessing predatory risk <sup>43</sup> .

- **Real-Time, Private Scanning:** Our solution works in **under a minute** and doesn't require sending the document to a third-party service. Many contract review services are either slow (hours or days if done by humans) or require internet connectivity to use cloud AI. LoanShark Sentinel operates **locally and swiftly** – a user can literally use their phone or laptop on the spot, even in the lender's office, and get results. The speed and offline capability not only make it convenient, but also build trust: the sensitive financial document isn't being uploaded to some server or saved – it's analyzed on the device and results are shown immediately. This approach resonates with user privacy concerns and practical constraints (e.g. a borrower might not have internet at a payday loan store) <sup>44</sup> . In the hackathon, we'll emphasize that our tech could even be packaged as an offline app eventually, which sets it apart from cloud-dependent AI solutions.
- **Clear "Risk Score" Metric and Education:** We introduce a **Predatory Risk Score** – a single metric that distills the loan's nastiness – which is novel compared to existing tools. For example, services like DoNotPay might identify a few issues in a contract but won't give an overall *"how bad is this loan?"* assessment <sup>45</sup> . Our score, combined with the "Top reasons" explanation list, gives users an actionable understanding of their situation. It's like a "credit score" for the loan's predatory nature – easy to grasp at a glance. Moreover, our focus on **educational explanations** (in plain language) is a differentiator. Rather than just labeling something "Arbitration clause – flagged", we say "Arbitration clause – this means you can't take the lender to court" <sup>19</sup> . This empowers users to learn and encourages better decisions. Competing tools often neglect the educational aspect; LoanShark Sentinel treats each highlight as an opportunity to inform the user, building trust and helping them avoid pitfalls beyond just this one document <sup>46</sup> .

In summary, **LoanShark Sentinel stands out** by being purpose-built for predatory loan detection, by combining rule-based clarity with ML smarts, by uniting legal and financial insights, by working instantly and offline, and by providing an easy-to-understand risk score that others lack. These differentiators position our MVP as an innovative solution in a space that has not seen such a tailored approach. Judges and users will recognize that we're not just doing "contract analysis" or "loan calculation" – we're solving a real problem in a novel way.

## Stretch Goals (Optional)

If time permits during the hackathon (or as future enhancements post-MVP), there are several extensions and refinements we could pursue to make LoanShark Sentinel even more powerful and user-friendly:

- **Expanded Clause Library & Localization:** Grow the library beyond the initial 30-40 patterns to cover more variations and edge cases. This could include state-specific predatory practices or even support for contracts in other languages (e.g. detecting Spanish-language loan terms that are predatory). We'd also like to integrate a knowledge of **state law limits** – for instance, knowing the legal APR cap in the user's state and flagging if the contract exceeds it, or recognizing if a choice-of-law clause picks a state deliberately to bypass protections. This would make the tool smarter and more locally relevant.
- **Interactive Document Highlighting UI:** Enhance the user interface to show the actual loan document with **highlights and tooltips** on each flagged clause. In the MVP we may only list the

issues, but a stretch goal is a PDF viewer or image overlay where the user can see each problematic clause highlighted in context, and hover or tap to read the explanation. This visual element would greatly improve user experience and understanding. It's technically feasible using PDF.js or similar libraries in a web app, and we'd aim to implement it given more time.

- **Mobile App Integration:** Create a simple mobile app or interface that uses the device camera to scan a physical contract and then runs the analysis. This would involve integrating the OCR and backend into a mobile-friendly form. The vision is someone at a loan store could snap a picture of the paper they're about to sign and get the LoanShark Sentinel report instantly. While likely outside the 8-day window to fully develop, we could prototype it or plan for it in a future iteration.
- **Advanced ML/NLP Enhancements:** In future iterations, we could explore more advanced NLP techniques to complement our regexes – for example, using a small transformer or BERT-based model (if it can be made lightweight) to catch nuanced or paraphrased predatory language that our patterns might miss. We could also train the ML model on a larger dataset (perhaps using unsupervised learning on a corpus of contracts to detect anomalies). Another idea is a **semi-supervised learning** approach: as the tool gets used, it could learn from new documents (with user feedback) to constantly improve its predictions.
- **Guidance & Alternatives for Users:** Going beyond just flagging problems, a stretch feature could be to offer **next-step guidance**. For example, if a loan is flagged as high risk, the app might suggest “Consider alternatives like local credit unions or payment extensions – predatory loans often lead to worse outcomes.” We could integrate links to consumer finance education resources or even build a small module that, given the loan's terms, calculates a comparison to a safer loan (like “If you borrowed \$500 on a credit card at 30% APR, you'd pay X instead of Y”). This moves the product from detection to also a bit of counseling, which could be very impactful.
- **API for Regulators or Lenders:** With more time, we could package our engine into an API or platform that could be used not just by borrowers, but by regulators or responsible lenders to audit loan terms. For instance, a state agency could run a batch of loan contracts through the API to identify patterns among lenders, or a forward-thinking lender could integrate the tool to ensure their loan terms aren't accidentally predatory. This could open up additional use cases and funding opportunities beyond the consumer-facing app.

Each of these stretch goals builds on the solid foundation of our MVP. They are modular enhancements – improving the clause library, the interface, the input method, the intelligence of the model, or the utility of the output. Depending on hackathon time, we might attempt a small step toward one of these (for example, implementing the PDF highlight view if resources allow, or adding one more language pattern). Regardless, they show a path for **sustained development** and how this project could grow from a hackathon demo into a fully-fledged product making a difference in the fight against predatory lending.