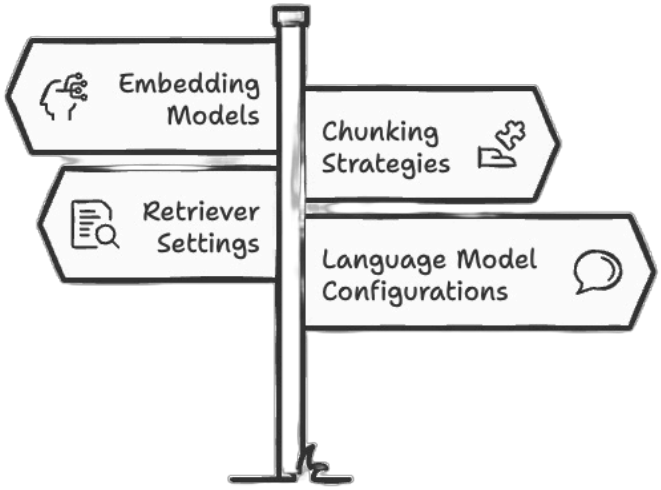
acumen

Fine-Tuning a Legal AI System with Genetic Algorithms: A Pragmatic Approach

In the world of legal AI systems, the quality of answers from a Retrieval-Augmented Generation (RAG) pipeline hinges on a delicate interplay of components and parameters.

Embedding models, chunking strategies, retriever settings and language model configurations all influence the relevance, coherence, and traceability of final outputs.





This presents a familiar optimization problem to any machine learning engineer: a vast combinatorial search space, unclear a priori what configurations yield the best results. Traditional manual tuning quickly becomes infeasible.



Genetic algorithms offer several practical advantages: they require no assumptions about the search space's smoothness or structure; they can handle mixed-type parameters (categorical and numerical) natively; and their population-based nature makes them well-suited for distributed evaluation. This aligns neatly with the needs of a modular RAG system, where the fitness of each configuration depends on the compound performance of many interlocking parts.

We'll walk through how we leveraged PyGAD, an open-source Python GA framework, to optimize our legal expert system's RAG pipeline. PyGAD stood out for its clean API, native support for mixed-type hyperparameters, and the ability to plug in custom fitness functions, which is critical for evaluating legal-answer quality. Its simplicity enabled rapid prototyping and its flexibility let us fine-tune with precision.

The Context: Legal Intelligence via RAG + LLM

Our goal: build an AI assistant capable of answering legal questions by extracting and reasoning over a corpus of legal PDFs.

Our initial setup follows a classic RAG pipeline:

1. ETL + Vectorization:
Legal PDFs and public legal information from various sources (and their metadata, footnotes, and structure) are ingested and chunked. Chunks are vectorized via an embedding model and stored in a vector database.

2. Query + Retrieval:
Incoming questions are vectorized and used to retrieve relevant text fragments.

3. Grounded Generation:
An LLM consumes the retrieved text and returns an answer—grounded in the source material.

4. Traceability:
Every retrieved text fragment is linked to rich metadata—including its source document, publication date, legal hierarchy (e.g. constitution > law > decree), and even location within the document. This allows the system to not only provide an answer, but also explain exactly where the information came from and under which legal context it applies. This is critical in legal domains where justification and provenance are as important as the answer itself.

The full setup includes sophisticated document hierarchies, footnote networks, and even the possibility of graph RAG. But even in a more conventional pipeline, hyperparameter tuning is critical.

The Optimization Challenge – Enter PyGAD

We defined an extensive parameter space, covering
both categorical and numerical options:

Categorical:

- Embedding Model (and varying dimensions in Matryoshka models)
- Text splitter
- Document Compressor (including rerank methods)
- Vector database retrieval settings
- Language model
- Compression Language model

Numerical:

- Chunk size
- Chunk overlap (percentage)

We tested over 12 embedding models x 5 text splitters x 4 document compression techniques x 20 retrieval settings x 18 language models x 10 compression language models x 5 chunk sizes x 2 chunk overlaps. **This resulted in over 8,500,000 possible combinations.**

Our goal: identify the configuration that maximizes the performance of our solution over a representative benchmark set of legal questions. With a well-configured test framework, it's easy to test and compare new generative AI methods, language models, and embedding models.

This systematic flexibility enabled us to confidently switch to GPT-4.1—because we could measure the performance uplift with hard metrics before making the move. (See the results at the end of the article.) In a rapidly evolving AI landscape, this kind of agile experimentation is not just useful—it’s essential.

We implemented a custom genetic algorithm using PyGAD, allowing us to encode configuration choices as chromosomes, evaluate fitness based on output quality, and iteratively evolve better solutions.

For each individual configuration, we ran the full RAG flow across 20 benchmark legal questions.



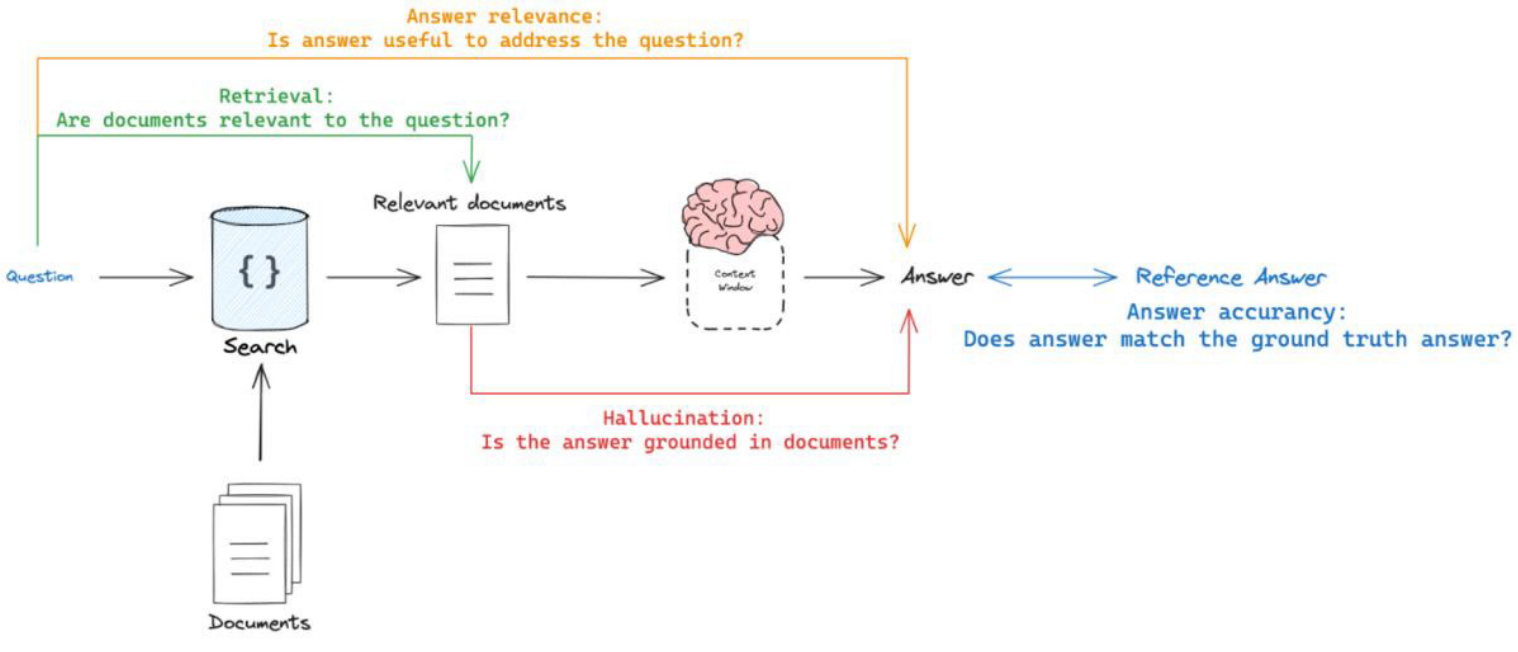
Fitness Function: Multi-Factor Legal Evaluation

We built a multi-metric evaluator, based on LangChain's definitions, to assess each configuration.

Metric	Weight for fitness	Description
Document Relevance	**30%**	Were the retrieved documents actually relevant?
Answer Faithfulness	**20%**	Did the answer stay grounded in the retrieved text?
Answer Relevance	**10%**	Did the answer meaningfully help with the legal query?
Answer Correctness	**40%**	Was the answer consistent with a known correct answer?

We weighed these into a single composite fitness score according to column "Weight for fitness". After numerous generations, the best configuration reached 91,4% fitness with a 95% answer correctness across the test set.

Below you can see LangChain's Evaluation Framework, that we used accordingly in our Fitness Function:



Notes on experimental setup

- During tuning, we used a fixed input of five parsed legal PDF documents (converted to markdown).
- Despite its complexity, the end-to-end pipeline (from document parsing to genetic optimization) was fully executable on a single local machine—demonstrating high efficiency and minimal resource overhead.
- For local development and testing, we used Chroma DB due to its lightweight setup and ease of integration. However, the final production-grade implementation relies on MongoDB Atlas Vector Search, providing scalability, resilience, and better observability.
- Due to the modular set up it becomes very easy to test a subset of parameters on their optimal value, while fixing other parameters.
- This framework allows us to easy add new parameters, like prompt strategies, context injection strategies, query re-writing, etc.
- Final configurations are revalidated against a broader document set later in our user acceptance testing environment.



Results & Learnings

While we'll publish full benchmark results soon, preliminary insights include:

Embedding models:

- Larger embedding models increase the document relevance metric and consequently improves overall fitness
- Cutting back on embedding dimensions—despite the matryoshka model design—led to a noticeable drop in performance, particularly in legal retrieval tasks where precision matters.

Chunking strategy:

- While exploring the parameter space with our genetic algorithm, we found no consistent trend in optimal chunk size across models.
- However, with the introduction of GPT-4.1, a clear pattern emerged—larger chunk sizes consistently delivered better results, suggesting the model's improved ability to handle and reason over extended context. We know that the GPT-4.1 model's can handle larger contexts but we haven't been able to establish if that's the key driver.

Compression techniques:

- We initialized our genetic search with three possible document compression techniques — LLMListWiseRerank, LLMChainFilter, LLMChainExtractor —as well as a baseline variation without any compression. Interestingly, the difference in overall fitness and performance across these options was negligible in our case.
- This suggests that the current generation of LLMs may already be powerful enough to internally filter and interpret the provided context in relation to the question.



Language models:

- We evaluated a wide range of language models, including lightweight local models via Ollama, various LLaMA 3 variants (such as 8B and 70B) served through Groq, and the full suite of OpenAI models.
- For our use case—text-based legal question answering—the OpenAI models consistently outperformed the others by a significant margin.

Text splitter:

- A significant amount of effort went into converting legal PDFs to markdown—a crucial step for reliable chunking. While there are still some intricacies to resolve in this process, we currently use a straightforward combination of CharacterTextSplitter and MarkdownTextSplitter to ensure structural consistency.
- As we continue to refine the markdown generation pipeline, we plan to explore Semantic Chunking using our existing Pydantic-based document structure, allowing for more contextually aware and legally meaningful segmentations.

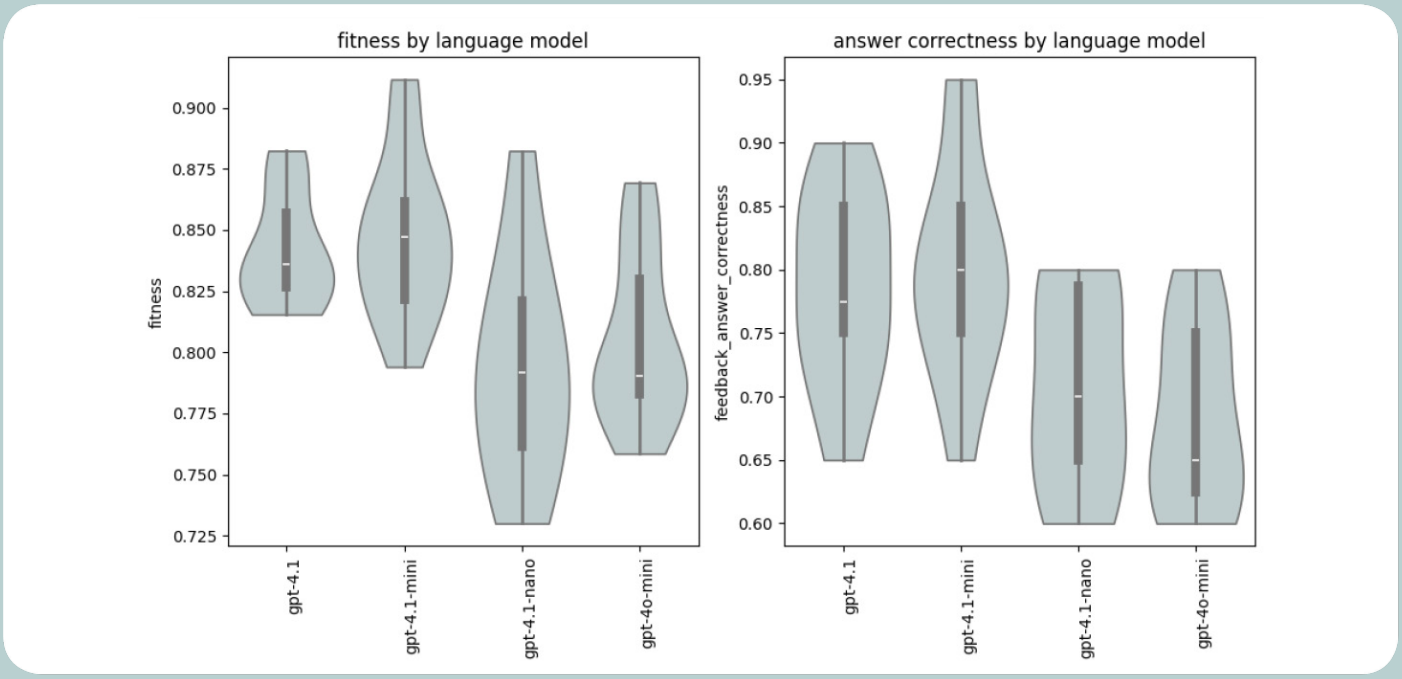


Update on the new GPT4.1 – models

Thanks to our modular, benchmark-driven setup, we were able to quickly retest the entire pipeline using GPT-4.1 as a drop-in replacement. Within hours, we had clear, empirical evidence: GPT-4.1 delivers significant improvements in overall accuracy & fitness.

This wasn't just a version bump—it was an upgrade backed by data. Because our pipeline is fully parameterized and test-driven, switching out a core component like the LLM is low-friction but high-impact. The performance gains were measurable across our full legal benchmark set, giving us the confidence to update our production configuration accordingly.

The result? Our clients are always running the best available version of the legal assistant, with changes grounded in real-world metrics, not guesswork. This agile experimentation loop ensures that each enhancement—be it a better model, new compression strategy, or optimized chunking—makes its way into the final solution only if it actually improves outcomes. That's the essence of our pragmatic, performance-first approach to legal AI.



These violin plots compare different GPT model variants on two key metrics: overall solution fitness (left) and answer correctness (right). The results clearly highlight the superior performance of GPT-4.1& GPT-4.1-mini, which shows both higher average fitness and greater correctness in legal answers. The distributions for GPT-4.1 are not only skewed higher but also tighter, indicating more consistent and reliable outputs across legal benchmarks—an essential trait in legal AI applications.

Building the AI-Powered Legal Assistant for Animo: Step by step

1) Project Kickoff & Requirements Analysis

Understanding Animo's needs to create an AI-driven legal assistant capable of answering legal questions, referencing case law, and providing grounded legal advice.

2) Data Collection & ETL Process

Ingesting legal PDFs, public legal information, and metadata, and structuring them for analysis through an ETL (Extract, Transform, Load) process.

3) Vectorization & Storage

All legal texts and structured information are transformed into vector embeddings and stored in a vector database for efficient retrieval.

4) Query & Retrieval

When a legal question is asked, it is vectorized and matched against the database to retrieve the most relevant text fragments.

5) Grounded Answer Generation

A Large Language Model (LLM) consumes the retrieved text and generates a context-aware legal response grounded in real legal documents.

6) Traceability & Source Verification

Each response is linked back to its source document, including publication date, legal hierarchy, and location in the text, ensuring full traceability.

7) Optimization with PyGAD

We employed PyGAD to fine-tune the AI model by testing over 8.5 million configurations using a genetic algorithm for optimal performance.

8) Multi-Factor Evaluation & Testing

We built a multi-metric evaluator to test the model on factors like document relevance, answer faithfulness, correctness, and legal applicability.

9) Implementation & User Acceptance Testing

The optimized model was tested in real-world scenarios, followed by broader user acceptance testing to ensure legal accuracy and reliability.

10) Continuous Improvement & Updates

The system was designed to easily integrate new models, including GPT-4.1, and continuously update configurations based on real-world performance data.

11) Deployment & Maintenance

The AI-driven assistant is now actively supporting Animo, delivering high-accuracy legal responses in seconds, with continuous support for improvements.

Summary:
How We Built Animo's AI-Powered Legal Assistant

At Acumen, we partnered with Animo to create an advanced AI-driven legal assistant. The goal was simple: help Animo's legal team find the right information faster and with more accuracy. Traditionally, searching through countless legal documents and PDFs was a time-consuming process. We aimed to change that.

First, we gathered all relevant legal documents, structured the information, and stored it efficiently so it could be quickly accessed. Using artificial intelligence, we taught the system to understand complex legal questions and find the most relevant answers in seconds.

The AI assistant doesn't just answer questions—it also provides full traceability, linking every answer back to its original source. This ensures that Animo's lawyers can trust the information, knowing exactly where it came from.

We also optimized the system to perform faster and more accurately by testing millions of configurations to find the best setup. Today, Animo's team is equipped with a powerful tool that saves time, reduces manual search work, and supports them in providing high-quality legal advice with just a few clicks.

This project reflects how modern technology, combined with the right expertise, can transform even the most traditional sectors like law.

Would you like to learn more about AI or our approach? Feel free to contact us at info@acumen.be

