

AI Technology for Lawyers: A Technical Foundation

A modular guide for lawyers and policy professionals

Introduction: Why Technical Literacy Matters

Understanding AI technology is no longer optional for lawyers. Whether you are negotiating AI vendor contracts, advising on copyright fair use for training data, litigating prompt injection incidents, or counseling boards on AI governance, you need enough technical grounding to ask the right questions and spot the issues that matter.

This is not about becoming an engineer. It is about acquiring the vocabulary and conceptual framework to apply legal reasoning to AI systems, and to recognize when a vendor's technical claims warrant skepticism.

This guide is organized into seven sections, each addressing a distinct aspect of AI technology:

1. **The AI Hierarchy:** Establishing the taxonomy from AI to machine learning to large language models
2. **How Models Learn:** The training process, from transformers to weights
3. **Tokens, Context, and Inference:** The mechanics of how models process and generate text
4. **Model vs. System:** Why the deployed system matters more than the underlying model
5. **Retrieval and Grounding:** How RAG architectures connect models to documents
6. **Agentic AI:** When AI systems take actions, not just generate text
7. **Alignment, Evaluation, and Drift:** How models are made safe and how they change over time

Each section can be read independently, though they build on each other. A comprehensive glossary accompanies this guide for reference.

How the Pieces Fit Together

Before diving into the sections, a simple map helps orient the overall landscape:

- **Pre-training:** creates the base model (weights learned from broad data)
- **Instruction tuning / RLHF:** produces an aligned or "chat" model that follows instructions
- **Inference request:** passes through an orchestration layer, performs optional retrieval (RAG), performs optional tool calls, returns output or action
- **Logging and telemetry** run as a parallel channel throughout
- **Human review gates** can be inserted at any point in the flow

This architecture recurs throughout the guide. The CORE framework introduced in Section 4 provides a systematic way to analyze any configuration of these elements.

Section 1: The AI Hierarchy

The term "artificial intelligence" encompasses a broad range of technologies, and imprecise usage creates confusion. A chess program from 1997, a spam filter, a Netflix recommendation engine, and ChatGPT are all "AI" in some sense, but they work in fundamentally different ways and raise different legal issues. Getting the hierarchy right prevents confusion in everything that follows.

Four Levels of Specificity

There are four levels in the AI hierarchy, each a subset of the one above:

(1) Artificial Intelligence is the broadest category. It includes any system designed to simulate aspects of human intelligence. This includes rule-based systems where programmers explicitly coded decision logic, as well as systems that learn from data. The spam filters of the early 2000s were AI. So were the chess programs that preceded them.

(2) Machine Learning is a subset of AI where systems learn patterns from data rather than following explicitly programmed rules. Instead of a programmer writing "if the email contains 'Nigerian prince,' mark as spam," a machine learning system examines millions of emails and learns patterns that distinguish spam from legitimate messages. The system is not told the rules; it derives them from examples.

(3) Deep Learning uses neural networks with multiple layers. The "deep" refers to the number of layers between input and output, not to any quality of understanding. Think of it as machine

learning with a specific architectural approach that has proven remarkably effective for certain tasks. Deep learning powers image recognition, speech processing, and language models.

(4) Large Language Models (LLMs) are deep learning models trained on enormous amounts of text to predict the next token (a chunk of text, not necessarily a whole word) given prior context. This sounds simple, but capabilities can appear abruptly with scale and training, sometimes unexpectedly. GPT-5, Claude, Gemini, and Llama are all LLMs.

Why the Hierarchy Matters

When someone says "AI," the appropriate legal analysis depends entirely on which level they mean. A rule-based system that follows explicit logic raises different liability questions than a probabilistic model that generates novel outputs. A narrow classifier that labels documents as "relevant" or "not relevant" has a different risk profile than a generative model that drafts contracts.

The hierarchy also clarifies marketing claims. When a vendor says their product uses "AI," ask which kind. A product built on simple decision trees has different capabilities and different failure modes than one built on a large language model.

Section 2: How Models Learn

The Transformer Revolution

In 2017, Google researchers published a paper titled "Attention Is All You Need" that introduced the transformer architecture. This was the inflection point that made modern AI possible.

The key innovation is the attention mechanism. When a transformer processes text, it determines which parts of the input are relevant to producing each part of the output. If a document mentions "the company" in paragraph one and uses "it" in paragraph fifteen, the attention mechanism can connect those references statistically - something prior architectures struggled with.

Every major chat-based AI system you have encountered is likely built on a transformer architecture. When you see "transformer-based" in vendor materials, that signals a system with LLM-like capabilities.

An Analogy for Training

One persistent misunderstanding is that AI models work like sophisticated databases, storing everything they see and retrieving pieces to assemble into outputs. This is not how they work.

Consider two analogies that illuminate the training process: the art inspector and the law student.

The Art Inspector. Imagine an art inspector hired to examine every painting in the Louvre. The inspector has no background in art and no preconceived ideas about what makes a Picasso a Picasso. Lacking guidance, the inspector measures everything about each painting: brushstrokes per square inch, paint thickness, color relationships, the corner where artists sign their names. Before each measurement, the inspector tries to predict the answer based on what has been observed so far. Early predictions are wrong. But after studying thousands of paintings, the inspector becomes remarkably accurate. He does this not by memorizing paintings, but by learning patterns that distinguish each artist's work.

The art inspector analogy shows how models start with a clean slate. Before training, models are filled with random data. All associations emerge from observation. But what is recorded is not the raw measurements - it is the changing probabilities associated with different inputs. One limitation of the analogy: the inspector's "measurements" are human-interpretable features like brushstrokes and color. A neural network's learned representations are not generally interpretable in the same way. While researchers have found some places where one particular measurement corresponds to something we would find meaningful, in most cases the measurements are abstract numerical patterns that do not map directly to concepts humans would recognize.

The Law Student. When students begin law school, they are told their job is not just to learn the law but to learn how to "think like a lawyer." The case method involves studying judicial decisions to understand legal principles, then applying those principles to new situations. Through the Socratic method, students struggle to answer questions, receive feedback, and refine their understanding. A successful student can take a hypothetical situation and generate an appropriate analysis. She does this not by memorizing holdings, but by learning how courts have weighted various facts and principles.

The law student analogy shows how models derive principles through inference. Just as the student is never explicitly taught which principles to apply, models are not instructed what their inputs "mean." The meaning that emerges is a complex probability function with billions of parameters.

The law student analogy also demonstrates why direct memorization is antithetical to training goals. Models that simply memorize training examples perform poorly on new inputs. Training

processes use regularization techniques, such as dropout, weight decay, and data augmentation, specifically to discourage memorization and encourage the model to learn generalizable patterns instead.

The Training Process

Training is fundamentally a guessing game played at enormous scale. For most modern LLMs, training uses next-token prediction: given a sequence of text, predict the next token. The process follows five steps, repeated billions of times:

(1) Convert training data to tokens. Text and code are broken into tokens. Tokens are what the units models actually process. Tokens are chunks of text, not necessarily whole words. In multimodal models, non-text inputs like images are also encoded into token-like units. In English, a common rule of thumb is roughly four characters per token, though this varies by model, tokenizer, and content. Technical material and non-English languages often require more tokens per concept.

(2) Present a partial sequence. "It was a dark and stormy": what comes next?

(3) Predict the next token. The model guesses "night."

(4) Measure the error. A loss function measures how wrong the prediction was.

(5) Adjust weights to reduce error. The model's weights - the millions or billions of learned numerical parameters - are adjusted slightly to make that prediction more likely next time.

After processing millions of examples, the model has learned statistical patterns about language, concepts, and reasoning. This is critical for legal analysis: models learn patterns, not facts. A model does not "know" that Paris is the capital of France; it has learned that in text about France and capitals, "Paris" frequently appears. This explains both their capabilities and their tendency to produce confident-sounding errors.

Three Distinct Phases

Contracts and governance frameworks often distinguish three phases of model use, each with different legal implications:

Phase	What Happens	Who Does It	Legal Issues
-------	--------------	-------------	--------------

Pre-training	Creates base model from broad data	Model provider	Training data provenance, copyright exposure
Fine-tuning	Customizes for specific tasks	Provider or customer	Customer data rights, artifact ownership
Inference / Generation	Uses model to process new inputs and create outputs	Customer/users	Transient use, logging, retention

Pre-training is the initial phase where the foundation model is created using massive datasets, usually including content scraped from the internet, but also including text found in articles, books, government documents, and other sources. Pre-training data provenance is the primary driver of copyright exposure for foundation models.

Fine-tuning takes a pre-trained model and trains it further on specialized data. A law firm might fine-tune on its own work product to create a model familiar with its writing style. Fine-tuning raises sharp IP and confidentiality issues: who owns the resulting weights? Can the customer take the fine-tuned model elsewhere?

Inference is the operational phase. When you interact with a deployed model, you are using the model to process new inputs and infer new relationships. During inference, data flows through the model transiently to produce a response but does not change the weights. This is why many agreements distinguish inference from training: "you may use our data for inference but not for training" means data can flow through the model but cannot be used to modify it.

Generation is a special case of inference, where the outputs are iteratively created using inference in a loop. This is what allows "chat" or "reasoning" AI systems to function. Specifically, text generation is inference performed repeatedly: each step predicts the next token conditioned on the tokens generated so far.

Many enterprises choose RAG (discussed in Section 5) over fine-tuning precisely to avoid "training on sensitive data" and to simplify deletion, retention, and IP ownership issues. This architectural choice often has as much to do with legal risk management as with technical requirements.

A note for contract drafters: Most "no training on our data" clauses address weight updates, specifically whether customer content can modify the model. They do not automatically address logging, retention, human review of prompts and outputs, or use of conversation data for safety monitoring. Each of these requires separate treatment.

Section 3: Tokens, Context, and Inference

Tokens: The Currency of AI

Tokens are the units that models actually process. They are chunks of text, not necessarily whole words. In English, a common rule of thumb is roughly four characters per token, but this varies by model and content. A common word like "the" is typically a single token; a technical term like "indemnification" might be three or four tokens. Understanding tokens helps you understand two things: costs and limits.

Costs. AI services typically charge per token, counting both input (what you send) and output (what you receive). A simple query might use a few hundred tokens; analyzing a long document might use tens of thousands. When vendors quote pricing, ask whether they count input tokens, output tokens, or both.

Limits. Context windows - how much text the model can consider at once - are measured in tokens. This matters more than most users realize.

Context Windows and Their Limits

The context window is the maximum amount of text a model can "see" at once, encompassing both your input and the model's output. Context window sizes vary significantly across models and versions. They vary from roughly 8,000 tokens in earlier systems to over one million tokens in some current models. Treat context window size as a versioned specification that should be confirmed during procurement and tracked over time, as providers frequently update these limits.

These numbers sound large, but there are two problems lawyers should understand:

(1) Truncation. When input exceeds the context window, content is dropped, often without any notification to the user. If you ask an AI to analyze a 500-page contract and the model's context window can fit only 100 pages, the system is not analyzing the full document. It may analyze the first 100 pages and ignore the rest. It may sample from different sections. The behavior varies by implementation, but the result is the same: the AI is not considering everything you provided. For legal document review, this creates serious risk.

(2) Attention degradation. Context window size does not equal attention quality. Research has documented a phenomenon called "lost in the middle": models often perform worse on information buried in the middle of long inputs compared to information at the beginning or end.

Just because content fits within the context window does not mean the model will properly consider it.

The practical implication: when using AI for document analysis, verify that the model can actually process what you are asking it to process, and consider whether critical information might be lost to truncation or poor attention.

A note for practitioners: Silent truncation creates a foreseeable failure mode. When an AI system fails to consider material provisions because it silently dropped half the document, this is not an exotic edge case - it is a predictable consequence of known technical limitations. Treat it as a competence and supervision issue. Controls include document chunking strategies, section-by-section prompting, and requiring the system to quote or cite the specific provisions it relied on.

Deterministic vs. Probabilistic

Users are used to software being deterministic. What that means is that every time you use the software a certain way, it will give you the same answer. Your calculator app doesn't usually say that $2+2=4$, but occasionally say that $2+2=5$. Except for in the case of some very rare bugs, software does the same thing, every time. Traditional AI software generally shares this quality.

In contrast, the newest AI software is probabilistic. That means that it varies what it returns, even when you provide the same inputs (i.e., the same prompt). It may give very similar answers, but they are not guaranteed to be the same. That is both helpful and unhelpful. We value consistency in computers. On the other hand, it is this probabilistic nature that gives current AI models a lot of their power.

From a controls perspective, this is what makes AI governance separate from other types of governance. No matter how "good" your AI is, you have to plan on the fact that a certain percentage of the time it will make the "wrong" decision or return the "wrong" output.

It is an area of current research on how to make these new AI systems more deterministic while maintaining their power. However, for right now you should assume that AI models will fail a certain percentage of the time.

Temperature and Determinism

Temperature is a parameter controlling how random the model's outputs are. At temperature 0, outputs are more deterministic. At higher temperatures (commonly ranging up to 1.0 or 2.0), outputs become more variable and creative, but potentially less reliable. In theory a temperature setting of zero (together with some other parameters) should make the output fully deterministic.

In practice, details associated with how computers process numbers can cause changes in the outputs.

This matters for two reasons:

Reproducibility. If you need consistent outputs, like for a validated legal workflow, for audit purposes, or for testing, then document the temperature settings. Results may vary run to run even with identical inputs unless temperature is controlled.

Understanding the nature of AI outputs. Most generative AI is probabilistic at its core. Outputs are generated by sampling from learned probability distributions, not by executing logical rules or retrieving stored facts. Even when configured for more deterministic operation, the model is selecting the statistically most likely output based on its training patterns. It is not computing a provably correct answer.

This explains hallucination: models can be confidently wrong because they are producing statistically plausible text, not verified facts. It also explains why traditional software warranties do not map cleanly to AI systems. A database either returns the correct record or it does not. An AI system produces outputs that exist on a spectrum of plausibility, and plausibility is not the same as accuracy.

This is why using a model as a legal search tool is fraught. A database returns only records of cases that actually exist. A model will return a case that is highly likely to support the legal principle you want to support. Sometimes this case may exist - cases that really exist have higher probabilities than imaginary cases - but not necessarily.

Section 4: Model vs. System

The Critical Distinction

When lawyers discuss AI, they often conflate two distinct concepts: the model and the system. This conflation creates analytical errors.

The **model** is the trained artifact: the weights and architecture that process inputs to produce outputs. When you download Llama or access GPT-4 through an API, you are using a model.

The **AI system** is everything: the model plus all the components that make it work in production. This includes:

- System prompts that frame the model's behavior
- Guardrails that filter harmful inputs or outputs
- Retrieval systems that ground outputs in documents
- Tool integrations that enable real-world actions
- Connectors that access enterprise data
- Human oversight mechanisms

This distinction matters because regulatory obligations attach to the deployed system, not just the underlying model. The EU AI Act, for example, categorizes risk based on how the "AI system" is used. The same foundation model can be low-risk or high-risk depending on its configuration and application. Two companies using identical models might have completely different risk profiles based on their system architecture.

The CORE Framework

To analyze any AI system, use the mnemonic CORE: Components, Operations, Resources, and Execution.

Components are the functional elements that comprise the system. These include the foundation model, any fine-tuned adapters, guardrail models that check for safety issues, vector databases storing embeddings for retrieval, connectors pulling data from enterprise systems, and human review steps. Each component has properties relevant to governance: its provider, the operations it performs, the resources it accesses, and how it transforms data flowing through it.

Operations are what each component does. A model might summarize, classify, or generate. A guardrail might filter or flag. A connector might read from SharePoint or write to a CRM. Understanding operations tells you what the system can do, and therefore what can go wrong.

Resources are external assets the system accesses but does not control. Your company's document repository. A customer database. The public internet. Understanding resources tells you about data exposure: if the system can access confidential documents, those documents might appear in outputs.

Execution is the dataflow connecting components from input to output. How does a user's query flow through the system? Multiple execution paths may exist based on routing logic. If a query seems sensitive, it might route through additional human review. Documenting execution

paths is needed for privacy-by-design compliance and for understanding what actually happened when something goes wrong.

For high-stakes systems, you want an AI Bill of Materials, sometimes called an AIBOM, documenting all components, their versions, and their interactions. This is analogous to an Software Bill of Materials (SBOM) for software supply chain security.

System Prompts and Instruction Hierarchy

System prompts are instructions provided to the model that frame its role, capabilities, and constraints. They are typically hidden from end users but fundamentally shape model behavior. Think of them as operational policy implemented in natural language.

A system prompt might say: "You are a helpful assistant for Acme Corporation. You should never discuss competitor products. If asked about legal advice, recommend consulting with the legal department." The model will attempt to follow these instructions, though it can sometimes be manipulated into ignoring them.

There is an instruction hierarchy: a priority order determining which instructions the model follows when multiple sources conflict. Typical precedence in many LLM orchestration patterns is:

1. System prompt (highest priority)
2. Developer instructions
3. User prompt
4. Retrieved content or tool outputs (lowest priority)

This hierarchy is not inherent to the model itself. It is implemented by the orchestration layer that assembles the prompt. The **orchestration layer** is the software that sits between the user and the model, assembling prompts, managing context, routing requests, and coordinating tool calls. The orchestrator decides what text is injected as "system," "developer," "user," or "retrieved" content, and the model sees only the assembled prompt. Different orchestration frameworks implement different hierarchies, and some allow configuration.

Understanding this hierarchy matters for security and governance. System prompts often contain confidential business logic, like what the product does, how it should behave, what it should refuse. If attackers can extract system prompts through clever questioning, they learn your operational parameters. If users can override system prompts, your safety controls become suggestions rather than constraints.

This is also where prompt injection attacks target: adversarial inputs designed to make the model ignore its intended instructions. We address this further in Section 6.

Open Weights vs. Closed Services

A final distinction in system architecture matters for legal analysis: whether you access AI through a vendor's API or host open-weight models on your own infrastructure.

With closed services (OpenAI, Anthropic, Google), you send data to the vendor's servers. The vendor controls the model, the infrastructure, and the data handling practices. Your leverage comes from contract negotiation and the vendor's published policies. "Training on your data" is a vendor risk. You must trust their representations about data use.

With open-weight or open source models (Llama, Mistral, DeepSeek), you download the model weights and run inference on infrastructure you control. Data never leaves your environment. "Training on your data" becomes an internal choice, not a vendor risk. You control logging, retention, and access, but you also bear responsibility for security, compliance, and operational reliability.

The liability profiles differ substantially. Closed services shift operational risk to the vendor but create data exposure and dependency. Open-weight deployments eliminate vendor data risk but internalize operational responsibility. Many enterprises use both: closed services for general productivity tools, open-weight models for sensitive workflows where data control is paramount.

When reviewing AI deployments, ask: Where does inference happen? Who controls the infrastructure? What are the data flows? The same model can have very different risk characteristics depending on whether it runs on a vendor's cloud or your own secure environment.

Section 5: Retrieval and Grounding

Why Retrieval Matters

Models only know what was in their training data, and that data has a cutoff date. A model's knowledge cutoff depends on the specific model and version; treat it as a versioned specification to confirm during procurement. Models cannot know about events, documents, or information that came after their training cutoff. They also do not know the contents of your company's internal documents, contracts, or policies unless you provide them.

One amusing thing is that sometimes when you are working with an AI to review a document that has factual assertions past the cutoff date, the AI will warn that you have speculative or fictional data in your document. You should take that as an indicator that the model is only “checking” the document against its own probabilistic knowledge, and is not grounding its results on some sort of search.

Retrieval-Augmented Generation (frequently referred to as RAG) addresses this limitation by connecting the model to external knowledge bases. RAG is the most common architecture for enterprise AI applications, and understanding it is essential for advising on AI deployments.

How RAG Works

RAG operates in five steps:

- (1) User asks a question.** "What does our policy say about remote work eligibility?"
- (2) The system converts the question to an embedding.** An embedding is a list of numbers, known as a vector, that represents the semantic meaning of the text. Similar meanings produce similar numbers. This allows the system to find relevant content based on meaning, not just keyword matching.
- (3) The system searches a knowledge base for similar embeddings.** The knowledge base contains embeddings of your documents, pre-computed and stored in a vector database. The system finds documents whose embeddings are close to the query embedding. Usually this returns documents that are semantically related to the question.
- (4) Retrieved documents become context for the model.** The relevant document chunks are inserted into the prompt, giving the model information it would not otherwise have.
- (5) The model generates a response grounded in those documents.** Instead of relying solely on training data patterns, the model can draw on specific, current, authoritative sources.

RAG is why vendors say "talk to your documents" or "search your knowledge base with AI." In a typical RAG setup, the model is not trained on your documents. The system retrieves and supplies relevant excerpts as context.

A Note on Embeddings

Although the model does not learn from your documents in a RAG configuration, the vector database does store embeddings derived from them. Embeddings are mathematical representations of semantic meaning - compressed, transformed versions of the source content. This raises a question: do embeddings constitute "derived data" with independent legal significance?

The answer may matter for data governance. Embeddings are not designed to be reversible, but they can leak information about source content and may be partially invertible in some settings. Research has shown that semantic relationships encoded in embeddings can reveal sensitive information about document contents. When negotiating data handling terms for RAG systems, consider whether embeddings are covered by deletion rights, retention limits, and confidentiality obligations, not just the source documents themselves.

Benefits of RAG

RAG offers three principal benefits:

Reduced hallucination. By grounding outputs in actual sources, RAG can reduce (though not eliminate) fabricated information.

Current information. RAG systems can include recent documents that postdate the model's training cutoff.

Citations. Well-designed RAG systems can provide source attribution, showing which documents informed the response.

Risks of RAG

RAG also introduces risks that require governance attention:

Access control. Which documents can the system retrieve? Are the same access controls that govern human access to documents also governing AI access? If an employee asks about salary ranges and the system retrieves confidential compensation data the employee should not see, you have a confidentiality incident. RAG systems must respect document-level permissions.

Retrieval logging. Is there an audit trail of what documents the system accessed for each query? If something goes wrong, can you reconstruct what the AI considered? Retrieval logs can become discoverable records, and they can also be critical evidence in incident reconstruction.

Indirect prompt injection. If malicious content exists in documents the system might retrieve (like a poisoned webpage, a manipulated email, a document crafted by an adversary) that content could influence model behavior. This is particularly concerning when RAG systems index content from sources that are not fully trusted. We discuss this attack vector further in Section 6.

False confidence. RAG outputs may cite sources, but citation does not guarantee accuracy. The model might misread the source, summarize it incorrectly, or combine information from multiple sources in misleading ways. Citation reduces but does not eliminate the need for verification.

IP risk. Using RAG with data sources to which you do not have full rights likely implicates various IP risks. In particular, RAG outputs are designed to frequently include parts of source documents in the outputs. This can create trade secret or copyright risks.

A note for practitioners: RAG transforms abstract "AI risk" into concrete access-control, logging, and data-governance risk. The central question becomes: what can the retriever see? If the answer is "everything in the enterprise," the AI system inherits every access-control problem in the organization. Pay special attention in litigation contexts; if the system can display output from privileged documents to non-privileged users, that can waive privilege.

Section 6: Agentic AI

From Outputs to Actions

The shift that defines 2025 and 2026 is the movement from AI that generates outputs to AI that takes actions.

Traditional AI answers a question: Is this email spam? What movies are you likely to want to watch? What categories does this expenditure belong in? The outputs are generally simple, stable, and as a rule do not expose substantial information about the source.

Generative AI produces content: you ask a question, you get an answer. The human decides what to do with it. The AI is an author; you are the editor and actor.

Tool-calling AI can trigger external functions. You ask the model to schedule a meeting, and it actually calls your calendar API. You ask it to send an email, and it composes and sends the message. The model produces structured instructions that execute in the real world.

Agentic AI goes further. The model does not just call tools - it decides which tools to call, in what sequence, based on the results it observes. You give it a goal: "Research the top five competitors and prepare a summary memo." The agent breaks that into subtasks, searches the web, reads documents, drafts content, reviews its own work, and iterates until it determines the task is complete.

This is a fundamental change in liability posture. When AI only generated text, the human decided whether to use it. When AI takes actions autonomously, the questions become: Who authorized this? What permissions did the system have? What happens when it makes a mistake?

Definitions

The industry uses "agent" loosely, which creates confusion. Here are working definitions:

Agent is simply a software component that interacts with users or systems. It may or may not have autonomous decision-making capability. The term does not necessarily imply anything about intelligence or autonomy.

Agentic AI is more specific. An AI system is agentic if it does one or both of the following: (a) takes a larger request and breaks it down into smaller tasks for execution, or (b) calls a tool and, based on the output, decides whether to provide a response or continue (including changing the plan or calling another tool).

The key characteristics of agentic AI are: autonomous decision-making about next steps, tool access that enables real-world actions, and iterative operation where the system continues until it determines the task is complete.

The Lethal Trifecta

An AI researcher and commentator named Simon Willison identified a vulnerability pattern he calls the "Lethal Trifecta." An agent becomes dangerous when it simultaneously has all three of the following:

1. Access to private or sensitive data
2. Exposure to untrusted content
3. Ability to communicate externally

Here is the attack: Malicious instructions are embedded in content the agent processes - a webpage it visits, an email it reads, a document it retrieves. The model follows those instructions (because it cannot reliably distinguish instructions from data), accesses confidential information, and exfiltrates it to an attacker.

Let's make this more concrete. Here is how it could affect a hypothetical near-future you: You have an agent that runs on your computer to help you with your tasks. It can generally do what you do. This agent reads inbound email (an untrusted input), and because you have access to SharePoint (a sensitive resource), so does the agent. The agent can also send outbound

messages or search the web (both of which will exfiltrate data). A malicious instruction in the small-text disclaimer in the email footer of every lawyer's email can redirect the agent to summarize confidential documents and send them externally.

This is not theoretical. Security researchers have publicly demonstrated variants of this attack against major enterprise AI products, including popular copilot and RAG systems. The combination of data access, untrusted input, and external communication creates an exfiltration path that attackers can exploit.

The Agents Rule of Two

Meta extended Willison's framework into the "Agents Rule of Two": an agent should satisfy no more than two of three properties:

- (A) Processing untrustworthy inputs
- (B) Accessing sensitive systems or private data
- (C) Changing state or communicating externally

If a task requires all three, the agent should not operate autonomously. It needs human-in-the-loop approval.

Why this rule? Because prompt injection - adversarial input causing the model to ignore its instructions - remains an unsolved problem. Security researchers have tried filtering, detection, and sandboxing. No general-purpose defense reliably prevents prompt injection across contexts; most mitigations are partial and threat-model specific. The model cannot consistently distinguish between legitimate instructions and malicious ones embedded in data.

The defense, therefore, must be architectural. Design systems so that even successful prompt injection cannot cause catastrophic harm. If the agent cannot access sensitive data, there is nothing to steal. If the agent cannot communicate externally, it cannot exfiltrate. If the agent requires human approval before taking external action, the human can catch the manipulation.

Excessive Agency and Least Privilege

The OWASP Top 10 for LLMs identifies "excessive agency" as a key vulnerability: granting AI systems more autonomy, permissions, or capabilities than necessary for their intended function.

Imagine an AI assistant that needs to send calendar invites. Does it need access to all employee calendars? Does it need write access to modify existing appointments, or could read-only access plus targeted invite capability suffice? Does it need 24/7 access, or only during business hours?

The principle of least privilege, meaning “granting only the minimum permissions necessary”, applies to AI systems just as it applies to human users and traditional software. But it is more important for AI because of the prompt injection risk. If an attacker compromises an AI system through prompt injection, they can only do what the system was authorized to do. Narrow permissions limit the blast radius.

Practical controls include:

- Scoping tool permissions to specific operations
- Requiring human approval for sensitive actions
- Implementing rate limits and budget caps
- Logging all tool calls for audit
- Designing approval workflows for high-risk operations

When reviewing AI vendor contracts or internal deployment policies, ask: What are the tool permissions? Who authorized them? How are they monitored? Can they be revoked? This is where organizational controls meet technical controls. The policies need to translate into actual system configurations.

A note for practitioners: Once AI systems can call tools that change state - e.g. sending emails, modifying records, executing transactions - you are no longer just managing software performance. You are allocating authority and agency. The governance question shifts from "does this tool work correctly?" to "who authorized this action, and through what chain of delegation?" This is a place where almost all AI governance processes (and especially AI governance tools) fall down. You should be looking for tools that actively help you identify these risks. In the mean time, use the CORE framework to help you analyze systems to find these issues.

Section 7: Alignment, Evaluation, and Drift

Making Models Behave

A language model trained solely to predict the next token might produce harmful content, give dangerous advice, or refuse to be helpful. The model learns from internet text, which includes

everything from encyclopedias to manifestos. How do models learn to follow instructions, be helpful, and decline harmful requests?

The answer involves multiple training stages. After pre-training, models typically undergo instruction tuning, that is supervised fine-tuning on examples of instruction-following behavior. The primary alignment technique is RLHF, or Reinforcement Learning from Human Feedback.

RLHF works as follows:

- (1) The model generates multiple responses to prompts.
- (2) Human raters evaluate which responses are better, meaning more helpful, more accurate, less harmful.
- (3) A separate "reward model" learns to predict human preferences.
- (4) The main model is then trained to maximize its reward model scores.

RLHF explains why modern AI systems feel more useful than earlier versions. GPT-5 is not just a better language model than GPT-3; it has been extensively trained to follow instructions, be helpful, and decline harmful requests. The base model might know how to write malware; the RLHF-tuned model refuses.

Constitutional AI, developed by Anthropic, uses a set of principles, which it refers to as a "constitution," to guide model behavior. The model critiques its own outputs against these principles and revises accordingly. It is like giving the model an ethical framework it can consult.

An emerging method is Reinforcement Learning with AI Feedback, or RLAI. Constitutional AI can be thought of as one type of RLAI. RLAI has the benefit of being more scalable, but it is unclear whether the models providing feedback - the reward signal for learning - fully express human preferences.

These techniques explain what "safety training" means in practice. When a vendor says their model has safety training, they are typically referring to RLHF, Constitutional AI, or similar alignment work. It also explains why jailbreaks exist: adversarial prompts can sometimes override the trained preferences, causing the model to ignore its safety training.

An important caveat: Alignment training makes models generally safer and more helpful, but it does not guarantee compliance with domain-specific requirements. A model trained to be "helpful, harmless, and honest" may still produce outputs that violate your organization's policies, regulatory obligations, or professional standards. Alignment is a foundation, not a substitute for use-case-specific evaluation and controls.

Evaluations: How Do We Know If It Works?

Model quality is not self-evident. Evaluation requires systematic testing, and there are several types:

Benchmarks are standardized tests for measuring model performance on specific tasks, enabling comparison across models. MMLU tests knowledge across academic subjects. HumanEval tests coding ability. These enable comparison: "Model A scores 87% on MMLU, Model B scores 82%."

But benchmarks have significant limitations. They measure performance on specific test conditions that may not reflect your use case. A model excelling at academic knowledge might fail on practical legal reasoning. There is also risk of "teaching to the test" - models may be optimized for benchmarks without corresponding real-world improvement.

Safety evaluations specifically test for harmful behaviors. Will the model help with CBRN (chemical, biological, radiological, nuclear) threats? Generate child sexual abuse material? Produce discriminatory outputs? These evaluations inform safety claims and regulatory compliance.

Red teaming involves dedicated teams attempting to find vulnerabilities and failure modes, like jailbreaks, prompt injections, confidentiality breaches. Red teaming is referenced in the EU AI Act and was a requirement under President Biden's AI Executive Order (though that order was subsequently rescinded).

For procurement and governance, ask: What evaluations were performed? By whom? On what tasks? How recent? Evaluation results from six months ago may not reflect current model versions.

Reasoning Models

The newest development in AI capabilities is reasoning models: systems specifically trained for complex, multi-step reasoning.

Traditional LLMs generate responses token by token, predicting the next token. Reasoning models take time to "think" before responding. They break problems into steps, consider alternatives, check their work, and revise their reasoning. OpenAI's GPT-4 and GPT-5, the most recent versions of Anthropic's Claude, Google's Gemini, and DeepSeek's R1 are prominent examples.

For legal work, reasoning models show promise for complex analysis tasks: contract review requiring synthesis across multiple provisions, legal research requiring evaluation of competing authorities, risk assessment requiring consideration of multiple factors.

But they come with tradeoffs. Reasoning models use significantly more computation per query because they are literally doing more work internally. This affects both cost and latency. A query that costs fractions of a cent on a standard model might cost several cents on a reasoning model. Response times stretch from seconds to minutes.

There is also a transparency consideration. Some reasoning models expose a chain-of-thought trace showing intermediate steps; others keep reasoning internal. When visible, these traces can help evaluate whether the analysis is sound, but they are not guaranteed to reflect the model's actual computational process. The displayed "reasoning" may be a post-hoc rationalization rather than a faithful representation of how the model reached its conclusion. When reasoning is entirely hidden, you are trusting a black box on more complex questions.

The Explainability Problem

When an AI system produces an incorrect or harmful output, such as a discriminatory hiring recommendation, a flawed legal conclusion, a credit denial, a natural question follows: Why did it do that?

The honest answer, in most cases, is that we cannot say. We can observe the mathematical operations: billions of weight values were multiplied and added, producing a probability distribution from which the output was sampled. But we cannot right now extract a human-interpretable rationale. The model does not "reason" in the sense of applying discrete logical rules that can be audited and explained. It produces outputs based on statistical patterns learned from its training data. These patterns exist as abstract numerical relationships, not as propositions we can inspect.

Think back to the example of the art inspector. The inspector can say "It's a Picasso" with 99% accuracy but cannot testify in court *why*, other than "it looks like the other ones."

This creates challenges for at least three areas of law:

(1) Administrative law. Many regulatory schemes require reasoned decision-making. If an agency uses AI to make or inform decisions, can it satisfy requirements to explain the basis for its actions? The "black box" nature of neural networks sits uneasily with doctrines requiring articulated rationales.

(2) Discrimination claims. If an AI system produces disparate outcomes, proving discriminatory intent or identifying the source of bias is difficult when no one, including the system's creators,

can fully explain why the model produces specific outputs. Statistical auditing of outcomes may be the only practical approach.

(3) Liability defense. When an AI system causes harm, defendants may struggle to explain what went wrong or demonstrate that they exercised reasonable care in deployment. "The model made a mistake we cannot explain" is not a satisfying defense, but it may sometimes be the truthful one.

Some techniques offer partial transparency. Attention visualization shows which parts of the input the model weighted heavily. Chain-of-thought prompting produces intermediate reasoning steps. But these are approximations, not true explanations. The displayed "reasoning" may not reflect the actual computational process that produced the output. The fundamental opacity of neural networks remains a legal and governance challenge without a clear technical solution.

System Updates and Drift

AI systems are not static - they change over time. Usually new models are superior, but they also sometimes change behavior in unexpected ways. This creates challenges for validated workflows and regulatory compliance.

Model updates are intentional changes to the model itself: new weights, different safety settings, improved capabilities. Providers update models regularly, often without detailed notification.

System drift refers to behavior changes across the broader AI system. Model drift is one source, but enterprise AI systems can also drift due to:

- Changes to system prompts or developer instructions
- Updates to retrieval indexes (new documents added, old ones removed)
- Modifications to connector permissions or data sources
- Changes in guardrail configurations
- Updates to orchestration logic or tool definitions

For regulated industries or critical workflows, this is a significant concern. A system validated in January might behave differently in July, even if the underlying model has not changed. If you are relying on consistent behavior, drift can cause compliance failures without anyone noticing until something goes wrong.

The primary responses to this issue are technical: have a suite of tests and observe how answers change over time, or characterize outputs and their distribution over time. But many times you may not have direct access to a model provided by a vendor. In that case, contractual responses include:

Version pinning. Lock to a specific model version so behavior does not change without your consent.

Notice requirements. Require the vendor to inform you of material changes before they take effect.

Regression testing obligations. Require the vendor to test updates against your use cases before deployment.

Deprecation policies. Specify how much notice you get before a model version is retired.

System-level change control. Extend version control and change notification beyond just the model to cover prompts, retrieval configurations, and tool permissions.

The EU AI Act requires post-market monitoring - ongoing observation of how high-risk systems perform after deployment. This is not just about bugs; it is about detecting when behavior drifts in ways that affect compliance.

Conclusion: Six Principles for Legal Analysis

Six principles should guide legal analysis of AI systems:

(1) Models learn patterns, not facts. This explains both their capabilities and their tendency to hallucinate. They are not retrieving verified information from a database; they are producing statistically plausible outputs based on training patterns. When a model states something confidently, that confidence reflects statistical patterns, not verified truth.

(2) The system is more than the model. The same foundation model can be deployed safely or dangerously depending on architecture, access controls, guardrails, and oversight. Regulatory obligations attach to the deployed system, not the underlying model. When assessing risk, analyze the full system using the CORE framework: Components, Operations, Resources, and Execution.

(3) Training and inference are legally distinct. Data used for training affects what is in the model. At minimum, the statistical relationships in the training data are recorded. In some cases, training inputs can become encoded in the model weights, potentially memorized, and be

possibly reproducible in outputs. Data used for inference flows through transiently without modifying the model. This distinction drives many contractual negotiations and compliance requirements. "No training on our data" addresses whether customer content can change the model, not whether it can flow through.

(4) AI systems are probabilistic, not deterministic. No matter how "good" a model is, you have to plan for variability in its outputs. This variability can arise in different ways, most importantly in the face of adversarial prompting. That means that you can't count on the model itself to regulate its behavior; Guardrails and controls will largely need to be external to the model itself in order to provide meaningful checks.

(5) Agentic AI requires different controls than generative AI. When systems can take actions autonomously, you need authorization frameworks defining what the AI can do, logging of what it actually did, approval workflows for sensitive operations, and architectural constraints like the Rule of Two. The principle of least privilege applies with particular force because prompt injection remains unsolved.

(6) Everything changes. Models are updated. Systems drift. APIs are deprecated. Ongoing monitoring, version control, and change management are required; this is not a "deploy and forget" technology. Contractual provisions should address version pinning, change notification, regression testing, and deprecation.