SIJILLA · R&D · ARTICLE

Law as Data Model: AI Beyond the Legal Assistant

November 2026 · Document 1.1 · sijilla.com/documents/rnd

AI in law is usually talked about as an assistant or an agent: it drafts, researches, fills in forms, tracks deadlines, summarizes case law, from specialized software to general-purpose AI bots. This layer is useful and practical, but for the substance of law and legal technology it is cosmetic. There is a second, fundamental layer, and it gets far less attention.

On that layer AI is the missing semantic layer: a translator between the natural language of law and formal data structures. Without this layer, law could not be made computable. With it, legal work can for the first time be built as data management rather than text production.

This layer is easy to overlook, yet lawyers run into its absence every day. Recall how it usually goes with an ordinary AI chatbot. You ask it to prepare a document or work through a question, and the first answer misses. With your second message you clarify what the legal issue here. With the third, the applicable law. Then you explain which facts matter, what format you need, and what even counts as a result. By the tenth message the answer finally comes together. Here is the thing: at that moment you were not "guiding" the AI toward the answer. You were assembling a data model by hand: the issue, the facts, the applicable rules, the form of the result. You were just doing it from the end. What this model is, and how to build it from the start, is what we take apart below.

Law as Code: An Old Dream

The idea of turning law into a "program" long predates computers. **In the ninth century, al-Khwarizmi, the man whose name gave us the word "algorithm," wrote his algebra in part for the problems of Islamic inheritance law:** he translated legal rules into equations and the division of an estate into an algorithm (al-Khwarizmi, Kitab al-Jabr; trans. Rosen, 1831, p. 86), moving the question of shares from the domain of legal argument into the domain of computation.

With the arrival of computers and the growth of computer science, the idea of programming law gained new relevance, owing to a striking number of parallels:

- A legal norm (conditions → consequence) ≈ if/elif → then;
- Ownership (possession, use, disposal) ≈ an Access Control List;
- A deal ≈ a transaction, and restitution ≈ a rollback;
- The hierarchy of norms and lex specialis ≈ rule priority and overriding;
- Successive versions of a statute ≈ version control; entry into force ≈ deploy;
- Registers (of legal entities, for example) ≈ state databases.

© 2026 Sijilla. All rights reserved.

Law is being programmed in practice, too: the Catala language, used to formalize the French tax code; the Rules as Code movement; Russia's official Concept for the Development of Machine-Readable Law Technologies; and other initiatives. Where a norm sits closest to arithmetic (taxes, benefits), this works.

However, beyond that boundary, every such project runs into the same limitation: **for code, "meaning" is a value; for law, it is sense**. Take the *Force Majeure* clause that sits in thousands of contracts. For code, the value is just text, identical to the letter, while law computes its meaning anew each time, depending on the applicable facts.

The Problem of Meaning

That difference is the wall. A value in code is bits with a type, and what they mean is fixed by construction. Sense in law depends on context, on the form of the record, and on the content itself: it is dynamic and many-sided.

Jurisprudence has known this problem for a long time. H.L.A. Hart called it the open texture of law: any rule written in natural language has a core of clear cases and a "penumbra of debatable ones", and it cannot be exhaustively formalized in principle (H.L.A. Hart, The Concept of Law (3rd ed., OUP 2012), ch. VII).

An example: Determine the meaning of the following elements and classify them: "*John*", "*01.10*", "*9.99*", "*Title*".

Code will honestly parse the types: string, date, number, string. But the semantic schema, what these things are and what they are for, has always been marked up by a human. A human reads differently:

- "*John*": a name and, probably, a party;
- "*01.10*": a moment in time, a date, a pointer to an event;
- "*9.99*": most likely a price; maybe a rate, or a 9.99% stake;
- "*Title*": a thing, a title, a bundle of rights attached to an object.

Put the elements into a single context and hypotheses appear at the level of legal relations: "*John*" is a party, "*9.99*" can be the contract price, "*01.10*" is a deadline, "*Title*" is what changes hands.

The Role of the LLM

The LLM is the first technology that builds semantic markup on its own, from context, rather than from a schema composed in advance by a human (Gilardi, Alizadeh & Kubli, ChatGPT Outperforms Crowd Workers for Text-Annotation Tasks, PNAS 2023). Before, for a system to understand that "*John*" in a document is a party rather than just a four-letter name, a human wrote annotation rules or labeled texts by hand. Now the model infers it from the document itself, roughly the way a lawyer does on a first read.

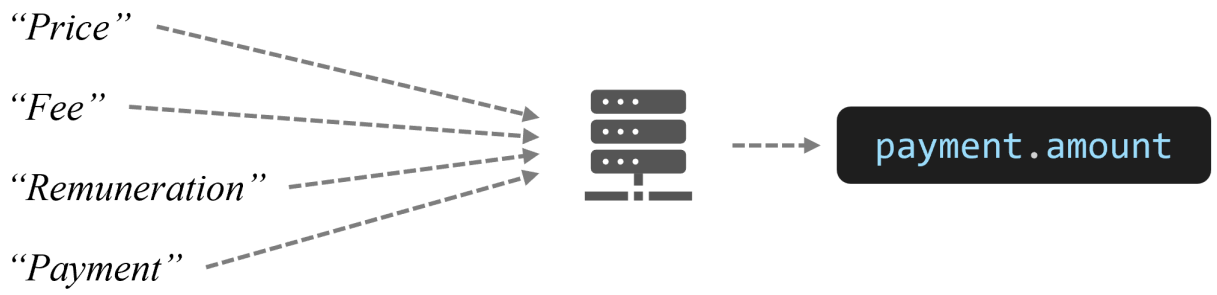
```
⚠ WARNING  meaning is not determined, only predicted. (W0001)
```

© 2026 Sijilla. All rights reserved.

The warning is correct. An LLM does not determine meaning, it proposes its most probable interpretation. The model does not store the meanings of words like a dictionary: it computes the probability of the next token from context, hence the errors and hallucinations (Dahl, Magesh, Suzgun & Ho, Large Legal Fictions, 16 J. Legal Analysis 64, 2024). They cannot be removed entirely, but they can be managed, and there are two answers here: an engineering one and a legal one.

The engineering answer: nobody leaves the LLM alone with the law. What stays deterministic is the data model itself, its fields, types and record lifecycle, along with the machinery around it: validation, deadlines, arithmetic, filling templates. All of that is computed by ordinary code. The semantic operations (reading, computing, writing) are performed by the semantic layer, but each of them starts in the model's fields and ends in them. Example: the LLM recognizes that "*John*" is a party to a legal relation; code then inserts him into the right slot of the contract or registers him as a party in the case management system. Between "understood" and "executed" an intermediate artifact appears, the markup itself: it can be opened, checked and corrected before the code does anything.

The legal answer is more interesting. Meaning in law carries a built-in tolerance for imprecision of form, and context is what closes that tolerance. One clause of a contract says "Price," another says "Fee," the annex says "Remuneration," and the invoice says "Payment": four words, one field, namely how much to pay. The contract header names a party "Supplier," the body calls it "Provider," and no court will read this as two different companies. The Romans put it as *falsa demonstratio non nocet*.



Four words, one field.

The semantic layer has the same tolerance, except that its limit differs on reading (input) and on writing (output).

On input the limit is set by sense: the form can be anything, as long as the value lands in its field of the model. The layer matches senses, not tokens, which is why "Supplier" and "Provider" turn out to be the same subject of the legal relation.

On output the limit is set by law. A semantic imprecision in a generated document that law can compensate for stays within tolerance; one it cannot is a defect. And the compensation does not need to be invented. Lawyers, like LLMs, make mistakes, hallucinate and distort meaning (the latter at times deliberately), and over the centuries law has built up mechanisms against such distortions: principle-level rules (good faith, prohibition of abuse of rights, and others), rules of interpretation (from the letter toward the parties' intent), presumptions for when data is lacking, review through the court hierarchy, alignment through case law.

© 2026 Sijilla. All rights reserved.

Approach: So the role of AI as the semantic layer is not to pin meaning down absolutely. Its task is humbler and more realistic: to determine meaning within the margin of error that law deems acceptable, and to place the result into the deterministic structure that code and data processing then work with.

Law as Data Model

So the machine has learned to work with meaning. But meaning has to be stored somewhere and retrieved from somewhere: a data schema is needed, that deterministic structure the semantic layer works with on input and on output. The good news is that lawyers and engineers do not have to design it from scratch. Such structures already exist; they have simply never been called data models.

Jurisprudence has several of them, and each legal tradition has its own. The most universal one lies in plain sight: the court judgment. Whatever the jurisdiction, its anatomy is the same: what has been established (facts), what applies (rules), what follows from this (conclusion). In civil law systems judgments these are the introductory, narrative, reasoning and operative parts. Academic and practice traditions polish the same construction for their own purposes. The German school trains lawyers through the Gutachtenstil (rule, definition, subsumption, conclusion). In the common law, students are taught to break a problem down by IRAC (Issue, Rule, Application, Conclusion), and practitioners reassemble it into CREAC, moving the conclusion up front.

The names of the parts change from jurisdiction to jurisdiction; the set of fields barely does: the forms differ, the slots are the same. These structures have been polished by centuries of practice. Nobody ever called them data models, but that is exactly how they work.

How does this carry over into LegalTech? The existing legal models are already fit to serve as the basis for data models: their elements are ready-made candidates for fields. Which model practice will converge on in ten years is pointless to predict, and unnecessary: there will be many formats even within a single firm or a single product. Litigators will run some flavor of CREAC; transactional practice will have its own set of fields, with Terms in place of Rules; insurers a third variant. This is not chaos but the normal life of data schemas, and they interconnect routinely, through field mapping, the way a CRM talks to a CLM. Whichever model you take, one thing matters: the basic elements and patterns have to be in place. And you do not have to look far for them; they lie in the legal models the profession has already polished.

Let us take these patterns apart on a reworked form of IRAC that I consider optimal for the semantic layer: **FIRAC**, or, to be precise, **F(p)IR(e)AC**: Facts (proofs), Issue, Rules (explanation), Analysis, Conclusion. Facts come with proofs, rules come with interpretation, and Analysis is understood as a computational step.

Facts and Rules. This structure has two independent primitives: facts (F) and rules (R), and they are interlinked: a fact pulls in rules, a rule issues a request for facts. An objection suggests itself here: what about the parties, the forum, the deadlines? All of these are facts. You can, of course, carve them out as separate sub-fields (parties, forum, deadlines) and in practice you will; but that is a matter of schema convenience, and it does not change their fundamental nature: they remain facts. For example, "Forum: arbitration in Singapore" is a fact that pulls in rules: procedural law, jurisdiction, the rules of

© 2026 Sijilla. All rights reserved.

the arbitral institution. The same logic applies on the rules side: statutes, precedents, regulations or contract terms can sit in sub-fields of their own, but they all remain rules. The remaining elements of the model are derivative.

(proofs). A fact has a lifecycle. An event becomes part of the model not on its own, but once it has been admitted into the case: linked to it, pleaded, contested, established. Every step of this cycle is governed by rules: the scope of proof, relevance, presumptions.

(explanation). Not every rule gives a clear instruction on how to apply the facts and what conclusion follows from their combination. Evaluative predicates often sit inside a rule's conditions and prescription: "reasonable time," "material breach." Their meaning is not stored in the norm; it is computed from the facts, precedent and enforcement practice.

Issue. Issue (I) is a query: a tension between a set of facts and the applicable rules. The query can be ex post, as in a dispute: the facts have already happened, and we look for the conclusion. And it can be ex ante, as in a deal: the desired conclusion is given, and we engineer the facts and rules that will lead to it.

Analysis. Analysis (A) is computation on the Issue's query, performed by matching facts against rules. The computation is semantic and is carried out by the semantic layer, but it takes its inputs from the model's fields and must return the result into the model's fields, together with a trace of the reasoning. The mechanics, meanwhile, are computed by ordinary code: deadlines, arithmetic, field validation.

Conclusion. Conclusion (C) is the result of the computation, recorded into the model. Its document packaging can be anything: a judgment, a memorandum, a contract, a demand letter, or part of a document, a recommendation, a contract section. And the conclusion can then return to the world and become a legal fact: performance of the contract starts a new cycle, this time as a fact.

Conclusion

In this optic, legal work turns out to be data management: extracting facts, selecting rules, computing conclusions, packaging the result into a form, be it a contract, a claim or a judgment. AI in this construction is not an author of documents but the semantic layer, a translator between human language and the data model. Which is why the reform is not that the machine now writes for the lawyer. It is that law, for the first time, has a working data model: the one operation it lacked through all the preceding decades was the semantic one, and that is the operation the LLM has closed.

The nearest application is already clear: the patterns taken apart above go into the architecture of LegalTech software. The semantic layer at the core, the model's fields (facts, rules, conclusions) in the foundation, from the first day of design, not on top of a finished product.

The further horizon is more interesting. Law as a data model will truly come into its own when LegalTech fully steps beyond software into the physical world. A camera in a port records cargo damage during loading, and the event lands in the model within seconds, not whenever a lawyer happens to learn of it: insurance, warranty, penalties and default all respond at once. The world begins to write into the model directly.

© 2026 Sijilla. All rights reserved.

And where the rules are deterministic, the cycle closes end to end: a child is born, and the shares in the estate recompute automatically. **Al-Khwarizmi began this calculation twelve centuries ago; what remains for us is to finish writing it.**

SOURCES

1. https://archive.org/details/algebraofmohamme00khuwrich/page/86/mode/2up
2. https://catala-lang.org/
3. https://oecd-opsi.org/publications/cracking-the-code/
4. https://pravo.ppt.ru/kontseptsiya/255728
5. https://www.oxfordlawtrove.com/display/10.1093/he/9780199644704.001.0001/he-9780199644704-chapter-7
6. https://doi.org/10.1073/pnas.2305016120
7. https://academic.oup.com/jla/article/16/1/64/7699227

© 2026 Sijilla. All rights reserved. Short quotations with attribution are allowed as the law permits; any other copying requires our written permission (sijilla.com/terms).