Electronic Design

DANIEL RHODIN, Founder and CEO, E-Sharp AB

Can AI Really Review a Schematic?

Generative AI can surface design errors that rule checkers miss, but it only becomes an engineering tool when every conclusion can be traced back to the schematic, the BOM, and the datasheet page it came from.

In one design, a digital isolator interface was powered from a 5-V supply while the bus connected to it operated with 3.3-V logic.

Nothing about the problem was particularly exotic. The signals simply didn't reach the logic level required for the isolator to interpret them reliably as high. The information needed to catch the mistake was available in the schematic and in the component datasheet. Nobody, however, had compared the right parameters before the board was ordered.

This is the frustrating reality behind many late-stage design defects. They are not always caused by difficult calculations or obscure physical effects. Often, they are ordinary mistakes that nobody had time to look for, or relationships hidden across schematics, bills of materials (BOMs), datasheets and design documentation.

That naturally leads to a question: Could AI help?

At first, generative AI appears to be almost perfectly suited to the problem. It can process large amounts of information, recognize patterns, and reason about relationships that may be spread across several documents. But there's an uncomfortable complication. A generative model is trained to produce an answer, even when the evidence available to it is incomplete.

In engineering, a plausible answer isn't enough.

The real question is therefore not whether AI can find a potential error — it's whether the engineer can verify why the AI reached its conclusion. A better-informed guess is still a guess. Only when the system can show its evidence can its conclusions be trusted as part of a design review.

AI-based design review isn't primarily a matter of finding a larger language model. It's a system architecture problem. The design must be represented correctly, the relevant evidence must be selected, and every technical statement must be traceable back to its source.

ERC and DRC Can't Verify Design Intent

Electronics development already includes several layers of automated verification.

An electrical rule check (ERC) can detect unconnected pins, conflicting drivers, and various inconsistencies in a schematic. A design rule check (DRC) is able to confirm that a PCB layout follows defined geometric and manufacturing constraints. These tools are indispensable, and it would be difficult to develop a modern electronic product without them.

However, ERC and DRC primarily check whether a design is consistent with a defined set of rules. They don't necessarily determine whether the design will do what the engineer intended.

A schematic can be electrically consistent while using the wrong component variant. A power rail can be connected correctly but still be unable to provide sufficient current. A signal could be connected to a valid input while operating in the wrong voltage domain. None of these situations necessarily creates an obvious rule violation.

This is why an experienced reviewer does more than check connectivity. The reviewer tries to understand the operating conditions, questions the assumptions behind the design, and compares the actual implementation with the datasheets and intended behavior of the product.

Design intent is difficult to reduce to a simple list of rules.

Human Limitation is Attention, Not Competence

When a design review misses an error, it's tempting to assume that somebody lacked the necessary knowledge. In my experience, that's often the wrong conclusion.

The real problem is scale.

No engineer can simultaneously keep every component variant, supply rail, pin function, signal level, package difference, footnote, and electrical limit active in memory. A modern electronic design simply contains too many

relationships.

Time pressure makes the problem worse. Attention naturally moves toward the parts of the design that appear most difficult or risky. Complicated power circuitry, high-speed interfaces, FPGA configurations, and critical analog signal chains receive scrutiny. Meanwhile, an ordinary voltage mismatch in a less conspicuous part of the schematic may pass unnoticed.

That isn't primarily a failure of competence. It's a limitation of human attention.

This is also where machines and engineers complement each other particularly well. A machine can apply the same level of attention to every relevant net, supply, pin, and component. A human is much better at evaluating ambiguity, understanding intent, and making difficult engineering judgments.

The opportunity isn't to replace the reviewer. It's to let the system perform the exhaustive work and bring the genuinely difficult cases to the engineer.

Why a General-Purpose Language Model Can't Review a PCB

A tempting approach is to export the schematic, attach the BOM and a collection of datasheets, and ask a language model a broad question: "Do you see any problems?"

This can work surprisingly well in small demonstrations. The model may notice an unusual connection, identify a missing pull-up resistor, or suggest areas that deserve further investigation. The problem appears when the same method is applied to a real commercial design.

We encountered three fundamental limitations:

Limitation 1: The Entire Design Doesn't Fit into a Context Window

A context window is the amount of information a language model can have available while producing a response. As that window fills with schematics, connectivity data, BOM information, manufacturing data, and component documentation, all of the material competes for the model's attention.

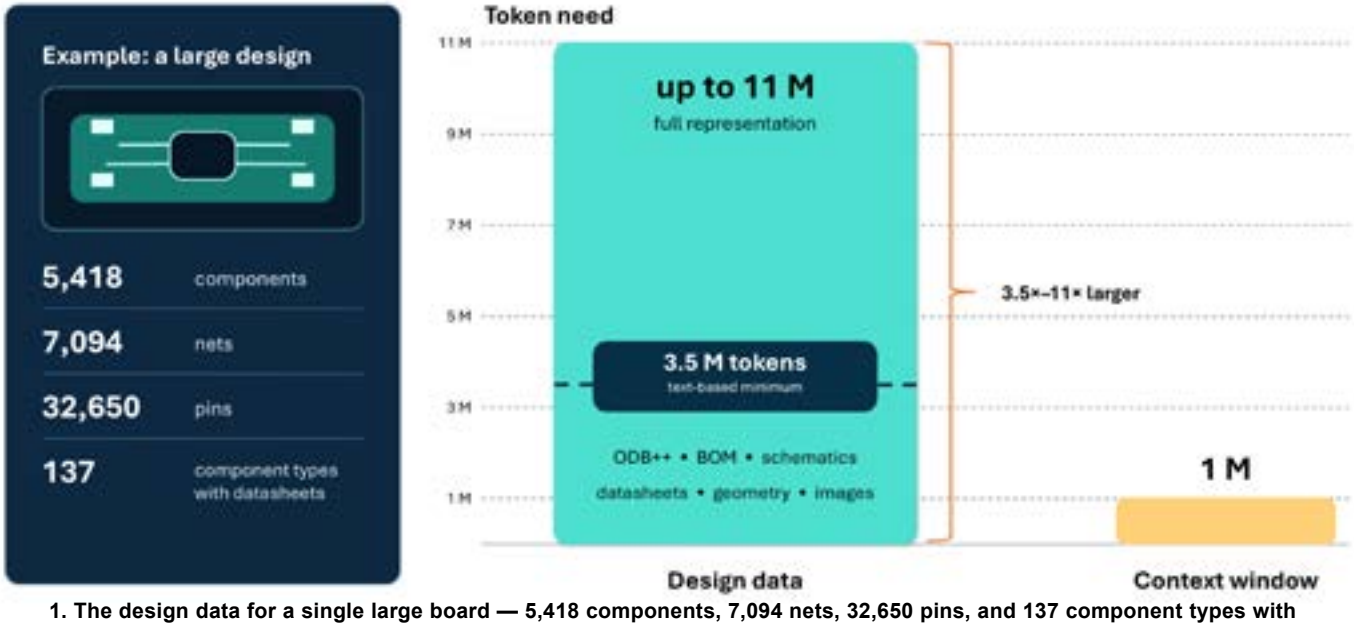
A larger electronic design can exceed even a one-million-token context window several times over *(Fig. 1)*. The model may need to consider thousands of nets and components, together with tens of thousands of pins, component properties, datasheet pages, tables, diagrams, and footnotes.

The obvious reaction is to look for a model with a larger context window. At the time of writing, however, that doesn't fully solve the problem.

The difficult part is rarely reading more data. The difficult part is identifying the right data.

A model given every schematic page and every datasheet must still determine which component variant, table row, test condition, package option, and footnote applies to the specific signal or power rail under review.

Putting more information into the prompt can sometimes make the answer worse. Relevant evidence becomes



1. The design data for a single large board — 5,418 components, 7,094 nets, 32,650 pins, and 137 component types with datasheets — runs from 3.5 million tokens as extracted text to 11 million for a full representation, 3X to 11X the capacity of a one-million-token context window.

surrounded by large quantities of unrelated but technically plausible information.

Limitation 2: Datasheets aren't Ordinary Text

A component datasheet isn't simply a collection of sentences. It's a structured technical document.

Meaning is often created by the relationship between a heading, a table, a row, a column, a unit, a test condition and a footnote. A value described as a maximum input voltage may be an absolute maximum rating, a recommended operating limit, or a condition that applies only to a particular package, variant, or temperature range.

Engineers interpret those relationships almost automatically. A language model doesn't necessarily see the page in the same way.

When a PDF is converted into text, the document may become a sequence of words and numbers. The extraction process can lose the original row and column structure. Once that structure disappears, the relationship between a value and the condition that makes it valid may also disappear.

The model can then find a number that genuinely appears in the datasheet and associate it with the wrong parameter. The resulting explanation may still sound completely convincing.

This is one of the most dangerous failure modes in technical AI. The source is real, the number is real, and the reasoning sounds reasonable. The relationship between them is simply wrong.

Therefore, making a datasheet searchable isn't sufficient. A review system must preserve or reconstruct its structure. It must be able to interpret the visual page when text extraction is ambiguous, and it must show the engineer exactly where the information came from.

Limitation 3: Generative Models are Optimized to Answer

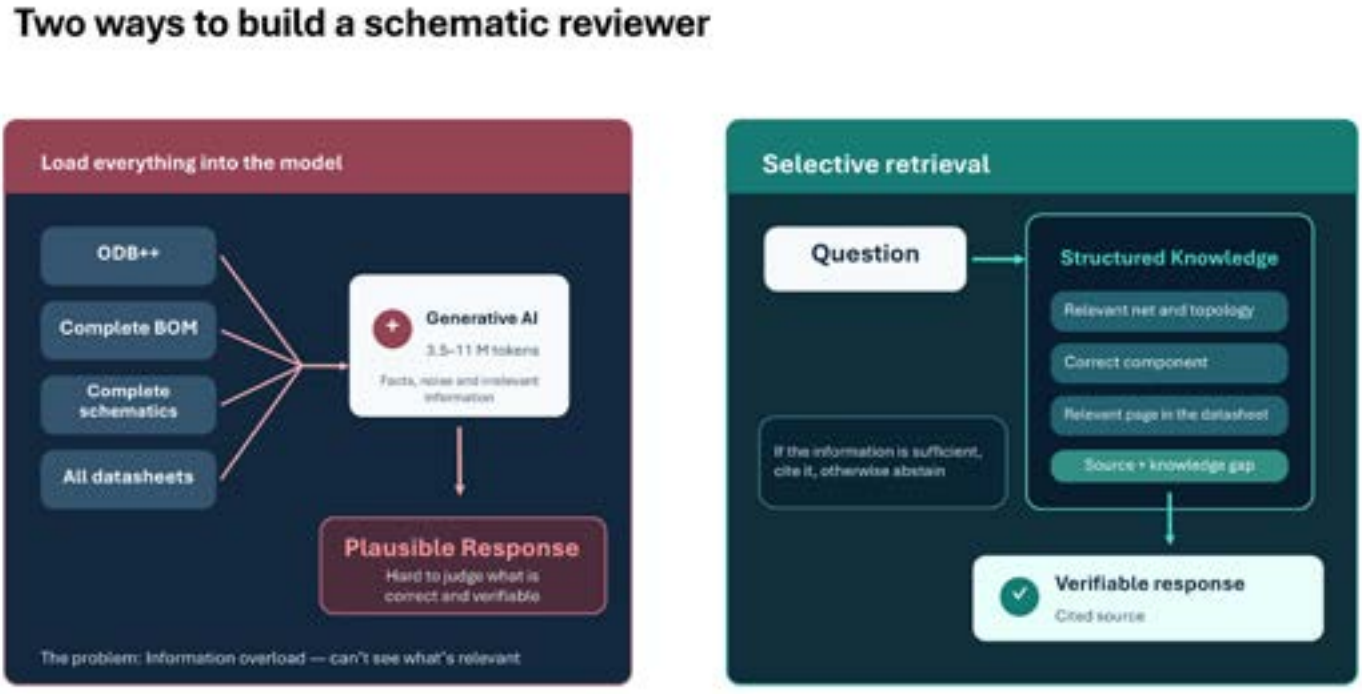
During our own work with AI-assisted design review, we repeatedly saw the same frustrating pattern. A model would provide an answer that looked entirely reasonable. The pin name was familiar, the electrical limit seemed plausible, and the explanation was technically coherent.

The problem was that the answer occasionally described the wrong component, package, or variant.

The model can be correct in many routine cases. Unfortunately, a design review is concerned precisely with the exceptions.

The exception may be a suffix that changes a component's startup behavior. It may be a pin assigned differently in another package. It may be an absolute maximum rating of 5.5 V rather than the 6 V used by a related product.

These details are easy to overlook precisely because the



2. Loading an entire design into a generative model yields a plausible answer that's hard to verify, whereas retrieving only the relevant design objects and datasheet passages gives the model a bounded problem in which every conclusion is tied to a cited source.

incorrect answer looks so familiar.

A language model carries a great deal of general knowledge about electronics. That knowledge is valuable when generating hypotheses and identifying relationships worth investigating. It must not, however, be treated as evidence about the specific design under review.

The review system must distinguish between what the model recognizes from previous training and what has actually been verified in the current schematic, BOM, and component documentation.

From a Document Pile to Reviewable Knowledge

This changed how we approached the problem when developing Probe. It's an electronics design review system that builds a structured representation of schematics, BOM data, manufacturing documentation, and component datasheets, and links every conclusion back to its supporting source.

Instead of treating the design as a large collection of files to be uploaded to a language model, the system creates a structured model of the design. Components, nets, pins, variants, and their relationships become defined objects that the system can examine.

This makes it possible to ask focused questions. Which components are connected to this voltage domain? Which devices are supplied by this rail? What orderable variant is specified in the BOM? Which datasheet applies to that exact variant? What electrical conditions are relevant to the pins connected to this net?

Datasheets are treated as engineering sources rather than as ordinary text documents. Finding a keyword or number isn't enough. The system must be able to identify the table, page, passage, and conditions that support the conclusion.

When the system performs a check, it retrieves only the relevant part of the design and the document passages required for that check *(Fig. 2)*. The language model receives a bounded engineering problem with defined objects and traceable evidence. It doesn't have to search blindly through the entire project, and it has less opportunity to fill missing information with general knowledge.

Cite or Abstain

The most important principle we arrived at is also one of the simplest: cite or abstain.

If a conclusion depends on a value from a datasheet, the system must show where that value came from. If the available evidence is insufficient, the system must not present an assumption as a verified fact.

There is an obvious objection to this situation. A model that can invent a value can invent a reference to go with it, so the principle is worth only as much as whatever enforces it.

The answer is that a citation isn't text written by the model. It's a reference that has to resolve. Every statement recorded against a design names a source type and a source reference, and the server dereferences that reference before it will accept the statement at all.

A connectivity claim must name a net, component, or pin that exists on this board. A schematic claim must name a token in the schematic that was actually ingested. A datasheet claim must name a part whose datasheet has actually been acquired. A reference that doesn't resolve is rejected, and the rejection names the remedy: fetch the datasheet before citing it. You can't point at a document that you never obtained.

The model's own reasoning is still admitted as evidence, but it sits at the bottom of the ladder and can't promote anything by itself. Raising a statement beyond a question takes independent sources of different types; the highest level takes a second reviewer, and the system refuses to let the author of an observation confirm it.

Agreement is treated the same way. Ten engineers concurring on a value leaves it at the tier that unsourced agreement earns; one manufacturer application note moves it two tiers higher. That's deliberate. Consensus is a weaker basis than a citation while looking stronger, and in a small team, a vote is never really anonymous. Making the cheap path to authority to "go and find the document" is the entire point.

This doesn't mean that the AI should avoid reasoning. Reasoning is necessary to connect schematic data, component properties, review rules, and documentation. However, the system must keep three things separate: facts observed directly in the design or its documentation, conclusions derived from those facts, and information that remains unknown.

That distinction can make the system look less impressive in a controlled demonstration. Instead of confidently answering every question, it will sometimes report that a component variant is unknown or that a datasheet condition can't be interpreted reliably.

In practical engineering, that restraint is a strength. A visible gap in knowledge could be investigated. An unsupported assumption presented as fact can quietly become part of the design.

Coverage Must Be Visible

A traditional review result often contains a list of findings. If the list is empty, it's natural to conclude that no problems were found.

But an empty list can mean two very different things. It can mean that the system examined the relevant parts of the design and found no issue. It can also mean that the system lacked the information required to perform the check.

Those two outcomes must not look the same.

Probe therefore describes review coverage as covered, partial, or blind. These terms don't describe whether the design is good or bad. They describe whether the system had sufficient evidence to perform a particular verification.

A power-supply check may be covered when the schematic, the exact orderable component variant, and the relevant datasheet conditions are all available. The same check may be partial when the component has been identified but a critical datasheet condition can't be interpreted with sufficient confidence. It may be blind when the system is unable to determine which component variant is actually fitted to the board.

Coverage makes the boundaries of the review visible. It tells the engineer not only what the system found, but what the system was genuinely able to examine.

A gap also carries its own history. When the system can't answer something, the record shows how often that same gap has been reached before — hit four times across three boards, never resolved. The fourth engineer to arrive at it can see that three people have already stepped around it. An abstention closes nothing and returns on the next board; a citation is bought once and closes the question for every board in its class. Making that asymmetry visible is what turns an honest gap from a dead end into a piece of work that's eventually done by someone.

Covering the Checklist isn't Covering the Risk

There's a second, harder question behind coverage, and it took us longer to face than the first. Reporting that every check ran, and that each had the evidence it needed, tells you the review executed. It says nothing about whether the set of checks covers the errors that actually cause respins. A board can be honestly and completely reviewed and still be blind to an entire class of defect, with nothing in the result to say so.

So, the review also reports coverage over error classes rather than over checks. Each class — voltage-domain mismatch, absolute-maximum violation, power budget, reset and sequencing, connector mis-mapping, thermal, and so on — declares the facts required to detect it at all, and the report states whether this board holds them.

Thermal is the uncomfortable one. The design data doesn't carry a thermal path, we don't model it, and we don't plan to. Thus, the class is reported blind on every board, permanently, so that nobody mistakes silence for coverage.

An engineer reading that knows to judge thermal somewhere else. An engineer reading a clean report that says nothing does not.

A Rule Hit isn't a Finding

Rules remain an important part of automated analysis. They're good at identifying defined patterns and directing attention toward potentially problematic areas.

A rule violation, however, isn't automatically a design defect.

During the review of one of our own boards, the system produced 16 rule hits. After further analysis, six relevant observations remained, four of which were genuine defects. Two of those defects were blocking issues.

One rule illustrates the pattern. It fired four times. Three of those flags turned out to be false positives. The fourth led to a real defect that could be confirmed using the design data and the component datasheet.

The rule hadn't described the complete problem. It simply directed the review toward the part of the design that required further analysis.

This distinction matters because a large collection of warnings isn't necessarily useful. Engineers already work with tools that produce more flags than anybody has time to investigate.

AI can add value by placing each rule hit into its technical context, retrieving the supporting evidence and helping determine whether the warning represents a genuine design issue.

False positives are manageable when the system can explain why they occurred. False confidence is far more dangerous.

What Does Practical AI-Based Design Review Require?

Our experience suggests that an AI-assisted design review system needs at least six things:

- It needs a domain model that represents components, nets, pins, variants and the relationships between them.
- It needs selective retrieval. The system should retrieve the design data and documentation relevant to the current question instead of sending everything to the language model at once.
- Its sources must retain enough structure for tables, conditions, units, and footnotes to be interpreted correctly.
- It needs a strict cite-or-abstain principle so that assumptions are not presented as verified facts.
- It needs visible coverage. The user must be able to see where the evidence is complete, where it's partial, and where the system is effectively blind.
- It must treat a rule hit as a starting point for analysis rather than as a finding in its own right.

These principles are part of the architecture behind Probe, but they are not unique to Probe. They're fundamental requirements for any AI system intended to support engineering decisions where a convincing but incorrect answer could lead to another PCB revision.

None of this removes the engineer from the review.

Machines are good at applying consistent attention across thousands of components, nets, pins, and parameters. Engineers remain better at understanding intent, resolving ambiguity, evaluating tradeoffs, and deciding whether a proposed solution makes sense in the context of the complete product. The purpose of AI-assisted review is therefore not to automate engineering judgment, but to give that judgment better evidence.

The Evidence Matters More Than the Answer

For engineering applications, it matters less whether an AI response sounds intelligent than whether its claims can be verified. The system must show where a value came from, distinguish one component variant from another, and state clearly when the available evidence is insufficient.

Consider the isolator again. Three numbers decided that board: the voltage on the isolator's supply pin, taken from the schematic; the minimum input-high level its datasheet specifies at that supply voltage; and the drive level of the bus on the other side. Every one of them was available before the design was released. Nobody had put them next to each other. Catching it didn't require insight. It required someone — or something — to retrieve three values and compare two of them, for that net and for every other net like it.

The most valuable result isn't always a discovered defect. It may be an exact datasheet reference, an identified gap in the design documentation, or a clear statement that the question can't yet be answered.

That's the point at which AI stops being an advanced guessing machine and becomes a useful engineering tool: Not when it always produces an answer, but when the engineer can see what that answer is based on.