A Case Study in Agentic Circuit Design for the Frugal Hardware Hacker

David R. Miller

{dave}@millermattson.com

May 17, 2026

This report asks whether an independent hardware developer who cannot afford frontier model API costs can use low-token-cost models in an agentic harness to design, simulate, and document a simple electronic circuit. I asked two low-cost LLMs, one locally hosted and one remote, to develop a hardware random bit generator where randomness is extracted from avalanche noise in a reversed-biased semiconductor junction, displaying one random bit on an LED each time a button is pressed. The models were prompted to simulate their circuit designs with ngspice and to generate schematics and parts lists. Both models failed to produce a working circuit design. Schematic drawing failed entirely despite remediation attempts with more capable models. I conclude that low-token-cost agentic AI does not offer meaningful assistance to the frugal hardware hacker. This reinforces a broader concern that capable agentic AI tools are out of reach for independent developers who have to buy tokens with grocery money.

1 The AI harness

After evaluating several open-source agentic harnesses, it became clear that Pi Coding Agent[1] has the lightest token overhead, which is crucial for our goal of minimizing token cost.

1 https://pi.dev/. I evaluated and rejected OpenClaw, OpenCode, and Hermes Agent for excessive token overhead.

2 The task and prompting strategy

The models were prompted to develop a circuit that displays a truly random bit on an LED each time a button is pressed, where true randomness is obtained from the avalanche noise in a reversed-biased semiconductor junction.[2] The models were allowed to use either a reversed-biased Zener diode or a transistor P-N junction.

To help ensure a working solution, the models were prompted to simulate their designs as SPICE models on a local installation of ngspice,[3] a mature and robust open-source analog and digital circuit simulator.

The models were also given command-line access to KiCad[4] which tightly integrates with ngspice and could facilitate schematic drawing.

2.1 Difficulties in this task

This electronic development task presents two notable challenges for the model:

- In general, LLMs don't seem to be well trained on using KiCad or ngspice on the command line, so the models had to discover how to interface with the tools, often by trial and error.
- Ngspice natively simulates a few noise sources (thermal, 1/f flicker, and shot noise) but does not natively simulate semiconductor avalanche noise. The agents were expected to propose a physical Zener diode or transistor and then simulate its avalanche noise in ngspice with a behavioral digital white noise source.

3 The models

I conducted the experiment with one locally hosted model and one remotely hosted.

3.1 Locally hosted

The locally hosted model had to work within the constraints of my modest home computer equipped with an NVIDIA RTX-5070ti GPU with 16GB VRAM, 32GB of DDR4 system RAM, and an old 12-core AMD CPU.

After experimenting with many open-weight locally hosted models, it became apparent that models that could fit in my home computer's memory would not be very adept at both reasoning

2 G. Guerrer, "RAVA: An Open Hardware True Random Number Generator Based on Avalanche Noise," *IEEE Access*, vol. 11, pp. 119568-119583, 2023, doi:10.1109/ACCESS.2023.3327325, https://ieeexplore.ieee.org/document/10295491.

3 https://ngspice.sourceforge.io/

4 https://www.kicad.org/discover/spice/

and tool use. For any hope of success, I would have to use the smartest model that my computer could host, which was Qwen3.6-27B.[5]

I was able to fit a Q6 quantization of Qwen3.6-27B into my computer by hosting it with open-source llama.cpp.[6] Of the model's 64 layers, 31 layers filled the 16GB of VRAM, and the other layers and KV cache occupied ∼18GB of system RAM. For the KV cache, I found that Q5_1 KV quantization was a sweet spot in the speed vs accuracy tradeoff.[7]

The model was not able to retain long-term coherence in Pi Coding Agent with a 65K-token context window. It required a context window no smaller than 128K tokens.

The result was a model that could reason, design, test, and debug autonomously all day, but at a snails pace of one or two output tokens per second.

Time is the cost of agentic development when subsidies and grocery money are insufficient.

The llama.cpp command line was:

```
llama-server \
  -m Qwen3.6-27B-UD-Q6_K_XL.gguf \
  --jinja --alias "qwen3.6-27B" --ctx-size 128000 \
  --no-mmproj-offload --no-mmap -ngl 30 \
  --presence-penalty 1.5 --temp 0.6 --top-p 0.95 --top-k 30 --min-p 0.0 \
  --chat-template-kwargs '{"enable_thinking": true, "preserve_thinking": true}' \
  --flash-attn on --cache-type-k q5_1 --cache-type-v q5_1
```

3.2 Remotely hosted

The other model, MiniMax-M2.7,[8] was accessed through OpenRouter,[9] a commercial API at low per-token cost of $1.50 per 1M output tokens. It is a 229B parameter model that performs nearly as well in benchmarks as models costing several times as much, making it a reasonable choice for our frugal developer.

4 Strategy and bootstrap

I first spent a few cents on OpenAI GPT-5.5, a high-token-cost frontier model, to form an implementation plan. I specifically asked GPT-5.5 to generate a plan that could be handed to a small local LLM for implementation. GPT-5.5 created an 880-line plan divided into eight different

5 https://huggingface.co/Qwen/Qwen3.6-27B
6 https://llama-cpp.com/
7 https://medium.com/@paul.ilvez/demystifying-llm-quantization-suffixes-what-q4-k-m-q8-0-and-q6-k-really-mean-0ec2770f17d3
8 https://huggingface.co/MiniMaxAI/MiniMax-M2.7
9 https://openrouter.ai/

files which could be given to a local LLM one at a time alongside two files containing a project overview and agent working rules. This helps keep the local LLM's context window focused on one thing at a time. The implementation stages were:

1. Set up the project environment, directory structure, package dependencies, etc.
2. Create the custom ngspice white noise node to simulate semiconductor avalanche noise and test it in ngspice.
3. Create and simulate the analog front end.
4. Create and test the circuit to threshold and sample the avalanche noise.
5. Create and test the LED output components.
6. Create and run automated testing of the entire circuit.
7. Create diagrams and a schematic.
8. Write the final project documentation.

5 Results

Both models worked very hard but failed to generate a valid circuit design.

Both models *appeared* to reason at length over every detail, e.g.:

> `The DC equilibrium: raw_bit (5V or 0V) connected to Clatch through Roff=1GΩ, and Clatch connected to ground through Rbleed=1GΩ. So the equilibrium is still (raw_bit + 0) / 2 = 2.5V when raw_bit=5V. But with a smaller capacitor, the initial transient charging to 2.5V happens much faster. Let me try Clatch=1nF instead of 1µF. (Qwen3.6-27B)`

However, both models reasoned about inappropriate matters. In the example above, the model was reasoning how to create a one-bit sample-and-hold latch using a capacitor and resistor. A capacitively leaky latch is entirely inappropriate for our circuit design.

Occasionally the locally hosted Qwen3.6-27B model worked for hours without making much progress as it ventured down rabbit holes that needlessly consumed tokens. The few times that happened, I was able to interrupt it and suggest an alternative path forward. The remote model, MiniMax-M2.7, was more adept at avoiding those rabbit holes.

Both models had to learn how to interface with ngspice, e.g.:

> `Actually, looking at the initial solution section, I see nodes like led0, led1, etc. So they exist in the circuit. The issue might be that ngspice-42 combines multiple .print tran statements into a single output with all signals. Let me check if there's a way to see what columns represent what. (Qwen3.6-27B)`

Each model wrote more than 1500 lines of Python code in several scripts to analyze the simulation output logs of ngspice. The locally hosted Qwen3.6-27B model, but not MiniMax-M2.7, also wrote Python code to generate various plots of the ngspice simulation output (Figure 1).

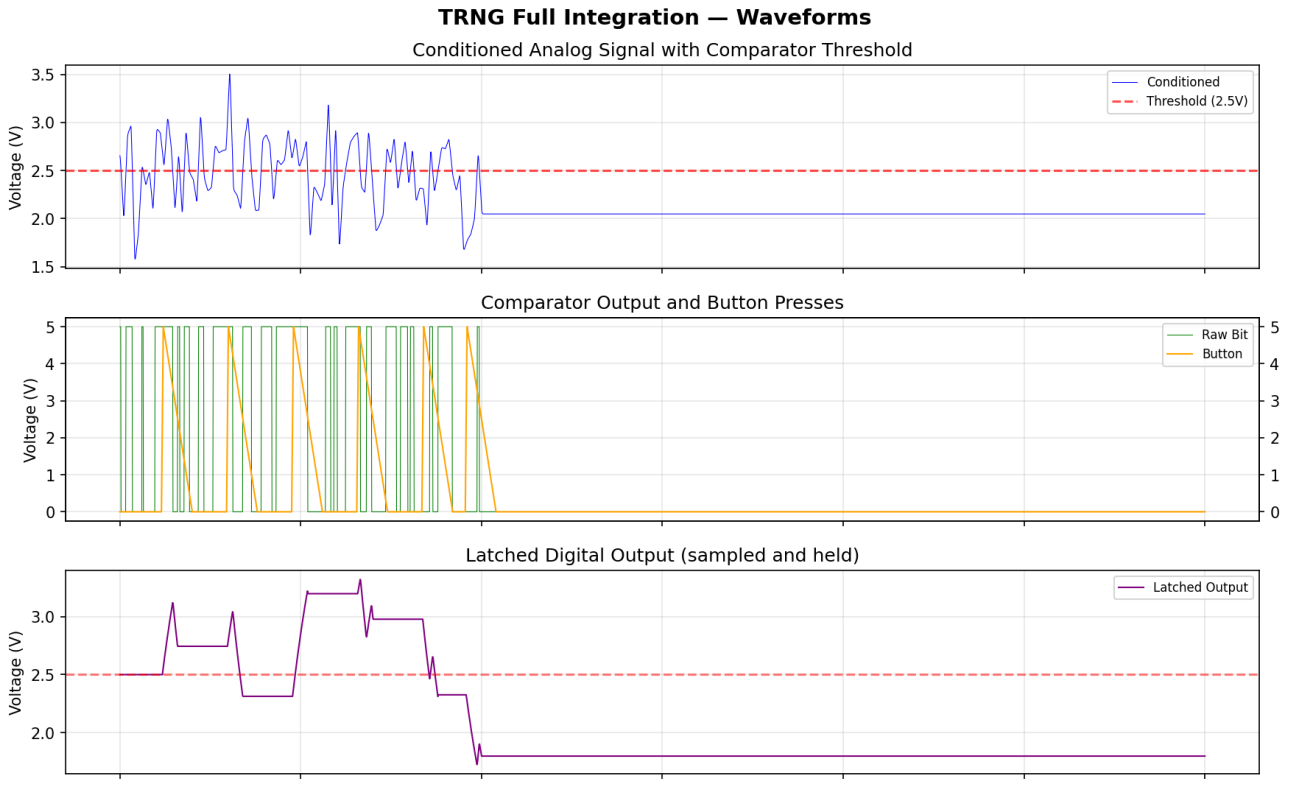

Figure 1: Qwen3.6-27B made pretty pictures for us from ngspice logs

However, the plots generated represented only an abstract behavioral simulation in ngspice and did not correspond to any circuit with specific components. Neither model was able to map the SPICE model to actual components.

6 Speed

The locally hosted Qwen3.6-27B model worked for 66 hours on this project with very little human supervision. The MiniMax-M2.7 model finished in ∼90 minutes (Table 1).

Table 1: Locally hosted Qwen3.6-27B was slow as molasses

Model	Wall clock time
Qwen3.6-27B	66 hours
MiniMax-M2.7	∼90 minutes

7 Costs

The cost of the locally hosted model was the marginal cost of electricity for 66 hours = ∼$0.60. We could also add the amortized cost of the computer, or at least the GPU, for 66 hours = ∼$1.30. The cost for remote execution of MiniMax-M2.7 was the token cost = $1.73 (Table 2).

Table 2: The more capable remote model cost a little less than the slow local model

Model	Cost (dollars)
Qwen3.6-27B (local)	$1.90
MiniMax-M2.7 (remote)	$1.73

8 The schematic problem

Neither model chose to emply KiCad for schematic drawing, instead choosing the Python Schemdraw[10] library.

When it became apparent that neither model could map a conceptual SPICE model to a schematic with specific parts, I manually intervened with remediation attempts.

For the locally hosted Qwen3.6-27B model, the last eight hours of its 66-hour run was responding to my additional prompts in an attempt to get it to produce a valid schematic.

For the remotely hosted MiniMax-M2.7 model, I switched temporarily to a more expensive and capable multimodal model, Kimi-K2.6,[11] for the last 30 minutes of its 90-minute run time in an attempt to prompt it toward a more successful outcome. Of the two models, it was the least bizarre. The failed result is shown in Figure 2.

10 https://pypi.org/project/schemdraw/
11 https://huggingface.co/moonshotai/Kimi-K2.6

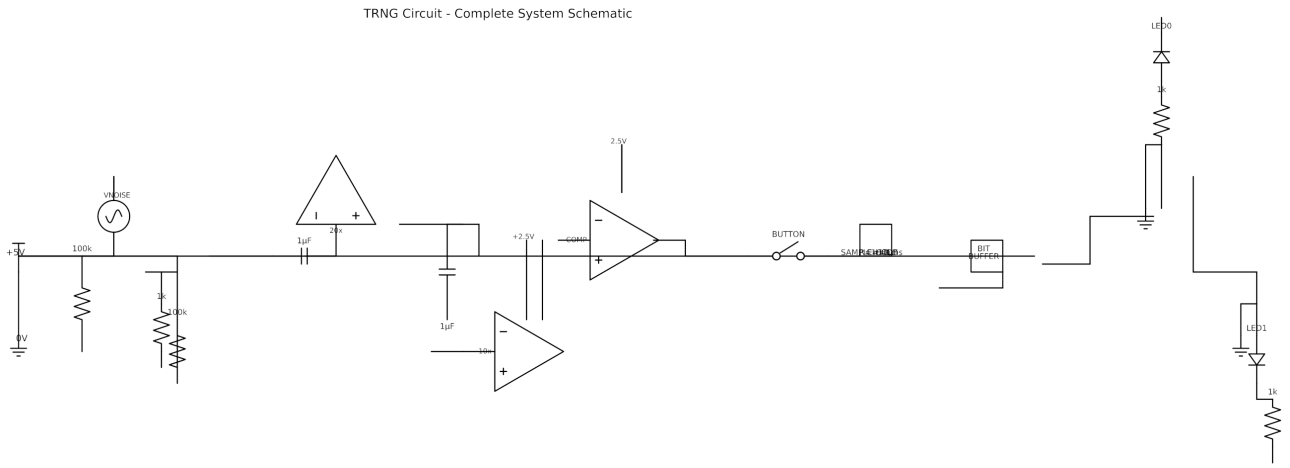

Figure 2: Uggh

9 Conclusion

This experiment was practical and personal: can a developer who cannot afford frontier model API costs use agentic AI for electronic circuit design? The answer is no, the technology is not there yet.

Both models tried hard but made poor design choices in their SPICE models and failed to produce an actionable schematic. They did, however, learn through trial and error how to drive ngspice from the command line. That's a minor success; neither model seemed to arrive well equipped to use SPICE models, but both figured it out, showing that agentic scaffolding could compensate, at least partially, for weaknesses in a model's training.

The greatest failure was mapping a conceptual SPICE model to a schematic drawing with specific parts. Even with additional manual intervention, neither model could produce a professional or functional schematic.

The local model worked for 66 hours of wall-clock time on dedicated hardware at under two tokens per second, with only part of the model fitting in GPU memory. The remotely hosted model was faster and nearly the same cost, and also failed. An expensive frontier model would likely have a better chance at success, but I've already used up this week's grocery money allowance.

This experiment supports the proposition that agentic AI, in its current form, is a tool of the rich.