Database Architects

A blog by and for database architects.

Saturday, November 7, 2026

Next-Level Benchmarking Overfitting

In my recent blog post I remarked that database benchmarking is in a crisis because AI agents start to overfit on individual benchmarks with questionable techniques, which drives execution times towards zero. This has happened with TPC-D in the past, and broke the benchmark. TPC-H tries to fight that with rules, but most people who publish (unofficial) TPC-H results simply ignore the rules. In combination with overly eager AI agents, this leads to brutal overfitting.

We are seeing the next level of that with the GenDB-V system. The corresponding paper commands "Stop Handcrafting Your Engine! GenDB-V: Towards AI-Generated and Formally Verified Database Systems", and reports speedups of 22× to 61× over Umbra and DuckDB, and all that in a fully verified system. These are indeed impressive claims, and would suggest that traditional database engines are in trouble.

Taking a step back, though, it should be clear that a performance improvement of a factor of 22 over Umbra is infeasible, at least if we are talking about traditional query processing. At SF10, the 22 TPC-H queries read roughly 15GB of (compressed) base data, and on a 9950X3D Umbra runs all queries in a total of roughly 540ms. On that machine the memory touched for data divided by the memory bandwidth (54GB/s) alone is about 280ms; anything below that is infeasible. Better compression might bring that down a bit, but not by an order of magnitude.

Unsurprisingly, the most dramatic gains of GenDB-V come from precomputation. The agents have carefully handcrafted auxiliary data structures for each query, which are computed at load time, and which avoid most of the work. The physical design is heavily tuned to the queries, too, for example, there is an extra structure that lists the `supplier` rows in `s_acctbal DESC` order, which makes `Q2` trivial. Similar per-query data structures are built for most queries. And we see that in the performance numbers: Several queries already run in near-constant time at SF100: Q2 in 0.7 ms, Q17 in about 15 ms and Q4 in about 17 ms on one thread. That is the TPC-D symptom, without stored aggregates.

There is also a "plain-array" mode, which, at first glance, is more fair, as it avoids these excessive preparations and also reports performance improvements of just roughly a factor of 3, which seems achievable. But when looking closely, we see that here the agents did something even more insane: They hard-coded their knowledge about the data generator. They "know" that each customer `k` will be at row `k-1`. And they know that dbgen numbers the `i`-th order `32·(i/8) + i%8`, so key `k` should be at row `8·(k/32) + k%32 - 1`. Which makes the join in query `Q18` super cheap.

How does all this fit together with the claim of a fully verified database engine? Well, the code *does* check if the hard-coded knowledge about the data generator holds in practice, e.g., by checking if the guessed join partner row is correct. If not, it falls back to a super slow generic nested loop implementation, but as that never happens with the official TPC-H data generator, that code path is irrelevant in the reported experiments.

Which brings me back to my original blog post claim: Letting AI agents optimize for a given database benchmark makes the benchmark useless. They will do their best to find insane tricks to speed up this particular benchmark, but none of that would work in a real system. The high tuning and synthesis time aside, deleting a single tuple (or

inserting one in the middle) can most likely cause the whole approach to fall off a cliff, as it forces the slow fallback path. Which is not something you want in a production system.

Thomas Neumann at 11:14 AM

Share

17 comments:

Anonymous November 9, 2026 at 11:03 PM

Stop handcrafting your engine. Start reverse-engineering dbgen!

Reply

Jiale Lao November 10, 2026 at 5:51 AM

Thank you, Thomas, for reading our paper and code so carefully. I am Jiale, one of the authors of GenDB-V. Your observations are correct, and I would like to add my view on the setting.

Two settings. Umbra and DuckDB are general-purpose engines: fast for any query on any data. GenDB-V, like Bespoke OLAP and JetStream, builds a specialized engine for one customer: their schema, data, hot query templates, and even hardware. Here, using knowledge of the workload is the purpose. For ad-hoc queries, such a system should hand over to the best general-purpose engines, such as Umbra and DuckDB. So I see the two approaches as partners, not competitors. I do not claim that agents can build an engine that is fast on many workloads: just for a very specific setting, specialized for the customer.

Verification is the main point. The key feature of GenDB-V is that instead of trusting agents for correctness, we can formally verify it independently. Every synthesized package is formally proved correct for every database of the schema. So each assumption must be checked in the code, and a failed check never gives a wrong answer silently: the code takes a general path. Today that path is GenDB-V's own slow proven code, but it can be easily extended to be Umbra or DuckDB.

Fair and unfair specialization. I agree that guessing where dbgen puts each key is fitting to the generator, not to a customer. A property that the storage creates, or that the customer declares and the system checks, is fair. An accident of the generator is not. I will constrain the agents to the first kind.

Agents also find good ideas. One example is TPC-H Q21 with agent-designed storage. The storage keeps each order's lineitems next to each other, for any input order. The EXISTS and NOT EXISTS conditions are then checked inside each order's own lineitems, in place of a semi-join and an anti-join on lineitem. No part of the query result is computed at load time. It is a classic physical-design idea that any engine could use.

I fully agree that the database community needs new kinds of benchmarks, and I made one attempt in this direction in my earlier work, SQLBarber (SIGMOD 2026). I am excited to see more great benchmarks in the future. Benchmarks drove the development of today's great engines, and they can do the same in the LLM era 🙂

Best,
Jiale

Reply

Replies

Thomas Neumann November 10, 2026 at 6:32 AM

You see I have two problems here: One is, you literally tell us to "stop handcrafting (our) engine". Which mean you are marketing your approach as a general solution and as a replacement for existing database engines, which it is clearly not. It might be an additional enhancement, like, e.g., physical design wizards did in the past. But you will always need a fallback engine, at the very least for ad-hoc queries.

And second, I am pretty sure that I could build a program that computes a secure hash over the input at load time to identify that it is the canonical TPC-H data, e.g., by hard-coding all relevant scale factors, and then produces the answers to all queries purely from the knowledge about the data generator, without ever looking at the data. That would have near zero runtime for all relevant scale factors. And it would still be "correct", because I could fall back to, e.g., an embedded DuckDB/Postgres/sqlite, the performance of which would then be totally irrelevant because it would never trigger in practice.

Which is clearly absurd, of course. But where do you draw the line? Why is recognizing the data set + scale factor "bad" when the other optimizations you do are not? TPC-H tries to codify this, which works to varying degrees _if_ people follow the rules (which many don't). I think we can only fight this if a) the benchmark is unknown to the engine developer before execution, and b) the data changes over time, which rules out the most extreme precomputations.

Anonymous November 10, 2026 at 6:38 AM

If formal verification is the main contribution, why are the headline results dominated by 20–60× speedups? Verification establishes correctness, but what does it tell us about the validity of those performance comparisons?

Jiale Lao November 10, 2026 at 7:49 AM

It is an honour to discuss this with you here. As a PhD student, I admire your work, and Umbra is one of my favourite systems and papers. I am sorry for the title: it describes a possible future as if it were a present fact. I will discuss changing it with my co-authors.

GenDB-V has two modes. Template synthesis builds specialized code for hot templates. That mode is an addition to a general-purpose engine, as physical design wizards were. Engine synthesis builds one general engine, proved correct for any schema, data and query in the supported SQL subset. In coverage and correctness it is a complete query engine for that subset. In performance it is not there yet: agents improved it over many versions, first on TPC-H and then on a second benchmark, but on manually created unseen queries it is still slower than Umbra. I will put its code and results in our repository.

My next step is an adversarial loop. Agents optimize the engine on all the benchmarks I have. Then the engine is frozen, and a separate adversarial agent writes new workloads against it: new queries on the known benchmarks, and completely new benchmarks. I report the frozen engine's result. After that, the agents may optimize on the queries where it lost, and the next round brings a new attack. The open question is whether the gap on unseen workloads shrinks over the rounds, that is, whether agents can build an engine that truly generalizes. This is in the spirit of your proposal of fresh workloads for each round.

On the line: your example is a good test. The proof checker ensures correctness for every possible database, and nothing more. Your program would not pass it. A hash has a fixed size, so different

databases can share a hash value; a secure hash is good enough in practice, but a mathematical proof must hold for all databases. Falling back to a production SQL engine cannot pass either, because none comes with proofs today; so currently GenDB-V falls back to its own engine, which is verified but slow. But I agree that the proof says nothing about performance: an optimization is correct for every database, and fast only for some. So the proof does not draw the whole line. Your (a) fits general-purpose engines; in the specialized setting the hot templates are known by design. Your (b) applies to both: a customer's data also changes, so the speed should hold on data the agent has never seen.

What I want to suggest is a direction: agents write code and proofs together, checked against a formal specification of SQL. Our proofs are only as good as our SQLSpec, which is our own reading of SQL. So I would like to invite the community to work on a shared, standard formal semantics for SQL. With it, verified engines and verified optimizations become possible for everyone, and maybe one day agents can bring a verified engine close to hand-built ones.

Reply

Anonymous November 10, 2026 at 4:48 PM

I don't think "correct SQLSpec" can solve the hardcoding dbgen logic problem Thomas pointed out. At the end of the day, you are verifying a database "correct", which has nothing to do with performance. In production, performance is part of correctness -- if a distribution drift of customer data causes a critical query to become orders of magnitude slower (as the example case would), that is a bug and an outage. The most egregious thing I find about this work is the attempt to portray formal verification as a green checkmark next to an arbitrary system, and disregard the fact that it proves a much narrower mathematical property that is highly sensitive to the exact formal definitions and assumptions with many pitfalls.

P.S. I generally like to assume the best of intentions from researchers, but I find your replies to sound a lot like AI.

Reply

Replies

Jiale Lao November 10, 2026 at 6:35 PM

Hi, I never claim that formal verification can ensure performance, both in the paper and in my previous responses. The ONLY thing formal verification can ensure is that, the results returned by the code, is allowed by the outcome specified in SQLSpec, and that is all.

For performance, we can only rely on benchmark feedback to guide the optimization, and as the final metric, not formal verification. Then we have the issue with benhmark evaluation: agents or systems overfitting to the benchmark, though in some cases, we want such specialization for customized code for hot templates.

PS: I give you my word, all the intentions from my previous comments come from me. I did use AI to re-write my draft, and the prompt is "make it simple, clear, and concise", and by Fable 5.2, and that is all. If we do not like the "AI-Style", from now on, I will use my original draft. Or if you want more, you can have a meeting with me to discuss more

Anonymous November 10, 2026 at 6:41 PM

You didn't even understand why the previous one said that "performance" should be part of "correctness".

Jiale Lao November 10, 2026 at 6:50 PM

All the optimizations for the hot paths, are checked at runtime: (1) If the data changes and the assumption does not hold anymore, the specialized code will never be executed. (2) If the fallback system is a powerful general-purpose engine, such as Umbra or DuckDB, this query would not be orders of magnitude slower.

Anonymous November 10, 2026 at 7:09 PM

Given all your clarifications, why is synthesizing a standalone database engine the right abstraction in the first place?

If the goal is hot-query specialization, existing engines already offer workload-aware physical design, materialization, query compilation, and other optimization techniques. Coding agents can also help engineers implement and test new optimizations within these engines.

What concrete capability does GenDB-V provide that cannot be achieved more economically by augmenting an existing engine? And what evidence demonstrates that advantage?

Anonymous November 10, 2026 at 7:38 PM

"All the optimizations for the hot paths, are checked at runtime: (1) If the data changes and the assumption does not hold anymore, the specialized code will never be executed. (2) If the fallback system is a powerful general-purpose engine, such as Umbra or DuckDB, this query would not be orders of magnitude slower."

Well I guess the point is. If your agent is overfitting too much, then the solution will be very sensitive to specific benchmark features. This means that, if the workload got drifted, even by a little bit, your solution will fail the checker and you can only fall back.
Ofc you can use a faster backend engine, but if your agent is falling-back in 99.9% of the time (ofc in more realistic, drifted workloads, not a fixed one), where is your contribution?
I feel one major point of previous discussions is that, you don't have systematic and reasonable enough evaluation, and people doubt you over-claiming.

Anonymous November 10, 2026 at 7:48 PM

“This means that, if the workload got drifted, even by a little bit, your solution will fail the checker and you can only fall back.”

Well, he can always add another "smart" agent to handle workload drift.

Jiale Lao November 10, 2026 at 7:56 PM

For hot-query specialization by coding agents, it is not a new claim from GenDB-V (but from previous GenDB, Bespoke OLAP, and JetStream), so I will not discuss more on this. I like the discussions in Bespoke OLAP paper: there are some advantages from directly implementing specialized code compared to tuning an existing system.

For the abstraction of synthesizing a standalone database engine, isn't this itself an interesting and exciting direction to explore, as a research project?

One important "economical benefit" of GenDB-V, I would say formal verification. No production SQL engine

is formally verified to the best of my knowledge (correct me if I am wrong). It was so difficult and expensive to formally verify something: seL4's 8,700-line C kernel took about 20 person-years! And now it is possible, to have a formally verified standalone database engine, and we achieve this in a much cheaper way, isn't this exciting?

BTW, just to give some contexts in case some people are not familiar with formal verification: formal verification does not mean "bug-free", just the code is verified against the formal specification, which is SQLSpec in GenDB-V. I do not claim this spec is correct. So I open-source it and hope the community could fix and refine it together.

Jiale Lao November 10, 2026 at 8:17 PM

"I feel one major point of previous discussions is that, you don't have systematic and reasonable enough evaluation, and people doubt you over-claiming."

I totally agree that I over-claimed for performance. The honest performance claim should be that, the synthesized engine can outperform untuned general-purpose engine, on a specific target .

I will update the claims. And I will run new experiments, with more strict settings and evaluations. Happy to take any suggestions. For example, the tool from Microsoft to generate skewed TPC-H data, and this is totally invisible to agents during synthesis?

BTW, I am always open to feedbacks, discussions, and suggestions, and will always improve based on these. Just hope, the discussion is open, healthy, friendly, and not emotional 🙂

Reply

Tobias Muehlbauer November 11, 2026 at 2:02 PM

This comment has been removed by the author.

Reply

Tobias Muehlbauer November 11, 2026 at 2:16 PM

Don't get me wrong, formal verification in form of model checking, SMT, TLA are all interesting and LLMs make it easier to define specs and run proofs. But then again, no formal verification will give you a bug free system in the real world. It's just yet another source we should tap into like sanitizers, code reviews, compiler warnings, unit tests, integration tests, production logs. Certainly great for a specific category of things where the spec is small and stable and a lot of investment can be justified to get to a trusted spec like peephole rules, atomicity/synchronization protocols, wire protocols. But even in the limited scope it's not bullet proof either, the blind spots remain: the spec itself, the translator, the tools, and in grand scheme of things, the OS, the hardware, the network, storage, ...

Reply

Anonymous November 11, 2026 at 2:18 PM

I think the broader question that "formal verification for systems" raises is, how do you formally verify non-functional requirements of a system?

Functional we now kind of know how to, durability maybe through effect systems (?), but performance is by far the hardest and I haven't seen any convincing approach there yet.

Reply



Enter Comment

View web version

Contributors

- Altan Birler
- Peter Boncz
- Thomas Neumann
- Viktor Leis

Powered by Blogger.