BI is back in its era of ferment

Emil Lindfors | September 22, 2026

Here is a question with two right answers. On the command line:

```
$ wren cube query --cube order_revenue --measures gross_revenue,net_revenue
  gross_revenue  net_revenue
         1672.0       1585.0
```

Net revenue, 1585.00. Now the same question in the chat UI, through my own MCP server, against the same DuckDB file: revenue is 1672.00. This is Jaffle Shop, the toy dataset every dbt tutorial uses. 100 customers, 99 orders, one quarter of 2018. I wrote both code paths, and neither of them has a SQL bug.

The difference is whether returned orders count. The rule that decides lives in a markdown file, `knowledge/rules/revenue.md`, and it says revenue means net here. The CLI loads that file. My server read the schema and stopped there, so it answered gross, and it sounded exactly as confident as the CLI did.

It gets better. `customers.customer_lifetime_value` also sums to 1672. Its column description says “across all completed orders”, and it sums every status, returns included. It is `SUM(orders.amount)` with extra steps. So a model that reaches for the obvious-looking revenue column gets gross, silently, and a model that reads the rule gets net. Two fields, both called revenue in conversation, disagreeing by design.

I spent August building a chat UI over a semantic layer and then arguing for the approach internally. This post tells that month twice. Once as a build log, and once through the lens I spent a PhD looking through, because the second reading is the one that changed what I would recommend to a customer.

The other side of the table

We build data platforms at work. That means we have watched, for years, how customers adopt new technology on the ingest and storage side, and the pattern is always the same: the technology that wins is the one whose interface gets standardised. Apache Iceberg is the current example. Once the table format is an open standard, the query engine on top of it becomes a choice instead of a marriage, and a customer who outgrows one engine moves to another with the data where it was.

So vendor lock-in is not something we like. We design for standardisation and modularity so our customers can switch across platforms if they need to, and we say so in the first meeting.

Now the same upheaval has arrived on the analytics side, and we are in the middle of it from the other side of the table. The question is no longer which engine reads the tables. It is how an agent should work with the data in the warehouse, and which of the twenty ways currently on offer will still exist in three years.

Dominant designs, and what happens when one cracks

The pattern has a name. Utterback & Abernathy (1975) described how a product line starts in a fluid phase, with many competing variants and rapid product innovation, and settles once one configuration becomes the accepted way to build the thing. After that, competition moves to process, cost and scale. The configuration that wins is the *dominant design*. The textbook example is the Model T: before 1908 you could buy a steam car, an electric car or a gasoline car, and afterwards you could buy a gasoline car with the controls where Ford put them.

Anderson & Tushman (1990) added the cycle that matters here. A technological discontinuity reopens the fluid phase. They call it an *era of ferment*: the old design and several new variants compete, nobody knows which will win, and then a new dominant design closes the era and incremental change resumes. Two of their findings are uncomfortable. The dominant design is rarely the technically best variant, and it is never the discontinuity's original form.

Business intelligence has had a dominant design for about fifteen years: a semantic model, a set of dashboards on top of it, and a per-seat licence. Power BI is its Model T. A Pro seat is $14 per user per month, Tableau Cloud Standard starts at $15, a Tableau Creator

seat is $75, and the products compete on exactly what Utterback said they would, which is price and process, not on what a dashboard is.

Agent-assisted generative BI is the discontinuity. Ask the question in a chat window, let the model write the query, get a chart back. Whether it is a *good* discontinuity is what the rest of this post is about. That it has reopened the fluid phase is not in doubt: Wren, dbt's semantic layer, Omni, Databricks, Snowflake, Microsoft's own MCP server for Power BI, and a dozen others are all shipping a different answer to the same question this year.

Theory makes three predictions about an era of ferment, and I have found all three useful:

1. **There will be more variants than survivors.** Most of what is being sold today will not be the dominant design, including the things that work.
2. **The winner will be picked by adoption, not by merit.** Arthur (1989) showed how increasing returns to adoption turn small early leads into lock-in by historical events, and that the locked-in technology need not be the best one. Seats, skills and the YAML your metrics are written in are all increasing returns.
3. **The safe bet during the ferment is the interface, not the product.** Iceberg was that bet on the storage side. The analytics side does not have its Iceberg yet, and I come back to that at the end.

The experiment

Innovation is inherently experimental, so it is hard to foresee the future, and the practical answer is to build something and watch what happens. Here is what I built.

The `knowledge/rules/revenue.md` file from the opening is a semantic layer, or the beginning of one. The idea: keep the definitions of your business terms (measures, dimensions, the join paths between tables, and the rules a schema has nowhere to put) in one version-controlled place, so everything downstream compiles its queries from there instead of carrying its own private idea of what revenue means. Business Objects patented the concept in 1991. Every BI tool since has shipped some version of it, usually called a model, a universe or a cube.

What's new is the consumer. A human analyst arrives already knowing that returns don't count and that April is a partial month. Nobody writes that down, because everyone senior enough to be asked already knows it. An LLM knows the schema and nothing else. So the conventions have to exist as files, or they don't exist at all.

The implementation I picked is Wren, from Canner. It bills itself as GenBI, generative BI: governed text-to-SQL over twenty-odd databases, where the model is a set of files you can review in a pull request instead of a paragraph buried in a prompt. The layer is called MDL, and `wren-core` underneath it is Rust on Apache DataFusion, so a question compiles against the model and then out into whichever dialect the target database speaks. I've

been driving DuckDB with it locally, and the same model would point at BigQuery or Snowflake without the measures changing.

There are two Wrens, and you want to know which one you're reading about. The one you may remember was a three-service Docker stack with its own chat UI, AGPL; the repos froze on 30 April 2026 and it's archived to `legacy/v1`. Wren today is `pip install wrenai`: a Python CLI and SDK, Apache-2.0 on the CLI and core, with an MCP server on the side. There's no UI in the open-source build, and that is exactly why it suits building on.

The project scaffolds `knowledge/` and `cubes/` empty. So a fresh checkout demonstrates the plumbing and nothing the semantic layer is for, and filling those two directories turned out to be most of the month.

The layer is five constructs, all YAML and markdown in git:

- **models** — a table plus everything the schema doesn't say. Column descriptions carry tagged facts: `[enum]` for what a code value means, `[unit]` for currency and whether `0` is a real zero, `[null]`, `[time]` for grain and actual coverage, and `[magic]` for real rows that look like placeholders (order 65 is `completed` with `amount = 0`).
- **relationships** — the join path, declared once, with cardinality.
- **views** — a join written once. A cube sits on exactly one base object, so without `customer_orders` no cube could slice revenue by anything about the buyer.
- **cubes** — named measures and dimensions.
- **knowledge** — rules, glossary, caveats, and curated NL→SQL pairs, as markdown.

There's also `.wren/memory`, a LanceDB index over the schema and the curated pairs. It's gitignored and rebuildable, so I don't count it as a construct.

The cube is the piece that does the enforcing:

```
name: order_revenue
base_object: orders

measures:
  - name: gross_revenue
    expression: SUM(amount)
  - name: net_revenue
    expression: SUM(CASE WHEN status IN ('returned', 'return_pending') THEN 0
ELSE amount END)
  - name: avg_order_value
    expression: SUM(amount) / NULLIF(COUNT(*), 0)

dimensions:
  - name: status
    expression: status

time_dimensions:
```

```
  - name: order_date
    expression: order_date
```

The return rule sits inside the expression, so everyone who asks for `net_revenue` gets it applied, and the grammar has no way to express an invalid join or an undefined aggregate.

Three more things in Jaffle Shop that the schema can't carry, all of which caught me before I wrote them down:

- **Denominators.** 38 of 100 customers never ordered. Average orders is 1.60 over the 62 who did and 0.99 over everyone. Repeat rate is 46.8% over 62 and 29% over 100. The rules file picks one and says which.
- **Enum meanings.** `return_pending` vs `returned` — which one has already cost you the money?
- **The clock.** Orders stop on 2018-04-09, so "last month" returns an empty table. The caveats file says to translate relative dates to absolute ones.

That's 99 rows and one quarter. Spider 2.0's average database has 812 columns.

Why not just let it write SQL?

Everyone I showed this to asked that, and it's a fair question. On the benchmarks people quote, the models are already good. Here's o1-preview on three of them, all three figures from the Spider 2.0 paper:

Benchmark	What it is	Score
Spider 1.0	academic schemas	91.2%
BIRD	semi-real	73.0%
Spider 2.0	enterprise	21.3%

Spider 2.0 is the one that looks like a customer. Its databases average 812 columns, often past a thousand; the gold SQL is frequently over 100 lines; the tasks are multi-dialect and multi-step. 632 of them. And the ceiling isn't moving quickly — the best score on the BIRD leaderboard is 81.95% from December 2025, against a 92.96% human baseline, and nothing has beaten it in the eight months since.

Attach one caveat to every number in this section, including the 96% ones somebody will quote at you. A CIDR 2026 paper re-checked the labels and found annotation error rates of 52.8% in BIRD Mini-Dev and 66.1% in Spider 2.0-Snow. More than half of the answer key is wrong. Correcting it moved leaderboard rankings by up to three positions across the five methods they re-scored; CHESS went from 4th to 1st.

In every study I could find, what closes the gap is the layer, not the model. Sequeda et al. (GRADES-NDA 2024) put GPT-4 on an enterprise insurance schema and went from 16% against the raw schema to 54% against a knowledge graph. dbt Labs (April 2026) took

Claude Sonnet 4.6 from 62.5% to 100% on in-scope questions with a semantic layer under it. Anthropic (June 2026) reported 21% to 95% of business queries automated in their internal analytics, which they attribute to "skills": the semantic layer plus canonical datasets, lineage, a query corpus and business context. That is the whole kit, not the layer alone, and the distinction matters when someone cites the 95% at you.

The finding I keep coming back to isn't in any of those headlines. In the dbt study, adding three more models took the layer to 98–100% in scope *and* lifted raw text-to-SQL to 84–90%, depending on which model you asked. Through the layer, which LLM you used barely mattered. The modelling mattered more than the model. (If you sell modelling, this is great news. If you sell models, less so.)

In that same dbt study the semantic layer scored 0% on out-of-scope questions, where raw text-to-SQL scored 70–100%. That reads as a loss until you look at how each one fails.

Join orders to line items before aggregating and a three-item order triples its revenue. The query runs. It returns a believable figure, off by about 30%, with a tidy summary paragraph explaining what it means. Nothing in the result indicates that anything happened.

The semantic layer's version of failing is `I can't answer that — churn_reason isn't in the model.` You extend the model. You don't spend a week debugging a number nobody caught.

For a user group that can read the generated SQL, silent fan-out is an annoyance. For the group I want to put this in front of, the ones who have been waiting three weeks for a dashboard, it's disqualifying.

A stack built to be swapped

If the theory is right that most of today's variants will lose, then the one design rule for building during the ferment is that every part has to be replaceable. Here is how far I got with that.

```
browser  →  chat-ui  →  mcp-wren-charts  →  wren CLI (MDL)  →  DuckDB
             (LLM)        (MCP tools)       semantic layer
```

Four containers, one `docker compose up`. The Next.js service holds the API key, so the browser never sees it, and the MCP server isn't published to the host at all. The model has no database credentials and no SQL escape hatch: everything it can do is a tool the MCP server exposes. Because it has no privileged access it's swappable behind one env var — Azure, OpenAI, Anthropic, OpenRouter, or a local vLLM or Ollama endpoint. It does need to be competent at multi-step tool use, since one answer is usually four or five calls.

Twelve tools: three for context (`get_instructions`, `get_knowledge`, `recall_context`), six for models and cubes (`list_models`, `describe_model`, `list_cubes`, `query_cube`, `run_sql`, `suggest_questions`),

and three for charts (`query_and_chart`, `get_chart`, `compose_dashboard`). The chart tools are the part Wren's own MCP server doesn't have, and they were the reason to write my own.

Charts come back as MCP Apps `ui://` resources, not images. assistant-ui mounts the widget in a sandboxed iframe and bridges its JSON-RPC calls back through an API route, so chart type, x and y stay switchable in place without re-running the query. Threads and messages live in IndexedDB, and a restored chart widget comes back live after a reload because the rows, the spec and the resource are all inside the persisted tool result.

Two datasets run in the same UI, with a switcher in the header and separate threads per dataset. Jaffle Shop over Wren, and about 1,800 Norwegian aquaculture sites over strata, which has weekly lice counts back to 2012 and is the one where the time dimensions and period-over-period comparison actually get exercised. strata calls a cube a metrics view; the MCP server translates at the boundary so both halves of the demo read the same. That boundary is the modularity in practice: the semantic layer underneath changed and nothing above it noticed.

Four things that broke

In roughly this order.

`wren memory` didn't run. `wren memory list|check|export|dump` all died with an ImportError. Those four commands read the LanceDB table through the `lance` Python bindings, which `wrenai[memory]` doesn't declare, so the extra installs and the commands still don't run. Adding `pylance` to the requirements fixed it.

Charts were PNGs in my UI and live widgets in Claude Desktop. SEP-1865 has you declare `_meta.ui.resourceUri` on the *tool*, which is what Claude Desktop reads. assistant-ui's AI SDK converter checks `output._meta.ui` and the flat `output._meta["ui/resourceUri"]` on the *result*, and never looks at the tool declaration. Both are defensible readings of the spec. Repeating the resource in both places is additive, so it works everywhere and nothing gets confused.

Capability detection stopped working under compose. The Streamable HTTP transport with `sessionIdGenerator: undefined` builds a new server instance per request, so the client capabilities from `initialize` aren't available at `tools/call` time. Auto-detection only ever works over stdio. The compose file sets `WREN_FORCE_UI=1` and stops guessing.

localStorage filled up. I had put thread history there. That was fine until two threads came to 26 KB, because every tool result embeds its full row set, and the ~5 MB cap was going to bite early. History moved to IndexedDB.

The field test, against the wrong build

On 18 August I pointed a fresh Claude session at the server, told it to go, and gave it no help. Six `query_and_chart` calls covering all five chart kinds, five `run_sql` calls including UNION ALL unpivots and scalar subqueries, two recalls. Every query succeeded on the first attempt, and the reconciliation checks it ran on its own confirmed the documented invariants to the cent.

The write-up it produced was more useful than the success rate, and it led with three problems: the cubes are documentation, not an interface; recall is read-only; and the knowledge files that the column descriptions cite can't be reached from any tool. I read that, agreed with all three, and started planning the work.

All three had shipped hours earlier. The host was talking to a server process started before the commit, and I had never restarted it. So I spent a while planning features I had already written. There is now a `server_info` tool: version, the tool list this build registers, and the MDL build time, so a session can check what it is connected to before it starts grading it. Add one before you run a field test, not after.

The feedback that survived the restart was smaller and mostly cheap:

- Results carry `row_count`, `truncated` and `elapsed_ms`.
- A failed query comes back with the engine's message plus a hint naming the tool that would have prevented it, so the agent corrects instead of retrying blind.
- Recall returns JSON with distance scores and repo-relative paths instead of a fixed-width table full of Windows paths.

The charting notes took longer and were the most useful thing in the report: one y column per chart became several, with stacked, grouped and 100%-normalized modes, a line-over-bars overlay on a second axis, currency and date formatting, sort and top-N, and deterministic jitter so 62 scatter points stop collapsing onto four vertical lines.

The one idea from it I still haven't built is letting a session mark a recalled NL→SQL pair as stale or wrong. Recall can be written to but not corrected. For a knowledge base that grows by accretion, that is the wrong way round.

Who gets to ask the second question

A dominant design also fixes who the product is for. Share of employees actually using BI tools: 19% in 1998, 22% in 2009, 35% in 2019, 25% in 2022. That's four figures from three firms with three methodologies, so don't read it as a series, and don't read it as a decline. Read it as flat, across twenty-five years in which both spend and capability went up a great deal. The seat prices above are the process-competition phase of the cycle doing its job, and the AI tiers land above the tiers people already don't use.

The queue is the structural reason. A business question ("is the Bergen site actually worse?") becomes a ticket, an analyst writes SQL, a dashboard ships, and the follow-up question goes back to the top. At Deputy, a VP of Customer Success asked for a dashboard

and was told it would take nine months, because that's how long it would take to build; every stakeholder group they had was unhappy with the data team. At incident.io, custom SQL fell from 70% of queries to 5% after they moved metric definitions into a shared model in Omni. No chatbot involved in that one.

What I think is changing is who gets to ask the second question. If a CEO can ask "is the Bergen site worse" and then ask the four follow-ups without filing a ticket for each, the analytics function shifts from producing charts to maintaining the definitions those charts agree on. I've seen that land hardest in smaller companies, where the C-suite is close enough to the data to want to dig themselves and there was never an analytics team big enough to absorb the queue.

That is also the reading Anderson & Tushman (1990) would push you towards. A discontinuity that changes who the user is tends to be competence-destroying for the incumbents, and the incumbents here are not the BI vendors, who will bolt a chat window onto anything. They are the analytics teams whose skill was turning a question into SQL.

What to do during the ferment

I spent my PhD on how an established industrial path gets transformed when a discontinuity meets actors who are able to act on it, in salmon farming rather than software (Lindfors, 2022). The mechanism transfers better than I expected. Four things follow from it for anyone building analytics on top of a warehouse right now.

Bet on the interface, not the product. Every vendor's semantic layer is its own YAML dialect today, so the expensive part of the work, writing down what your company means by revenue, is locked to whoever you wrote it for. That is Arthur's lock-in in its purest form, and the way out is the same one that worked for storage. Open Semantic Interchange is the attempt: one vendor-neutral spec for metrics, dimensions and relationships, so a BI tool, a query engine and an agent can read the same definition. It was announced in September 2025 by Salesforce, Snowflake and dbt Labs among others, published a v0.1 spec on 27 January 2026, was accepted into the Apache Incubator on 10 July as Apache Ossie (renamed to stop colliding with every other OSI), and has grown from 17 organisations to over 50. Databricks and Dremio are in it too. It is not Iceberg yet. Wren's MDL isn't an Ossie format, and moving would be a conversion, not a copy, so don't plan on models travelling between vendors unchanged today.

Keep every module replaceable, and know which one you are locked to. The model behind an env var, the semantic layer as files in git, the MCP server as the boundary that translates between two layers. You will still be locked to something. Choose what, on purpose, and write it down.

Run the experiment with real users, and expect to throw it away. Abernathy & Clark (1985) mapped innovations by whether they preserve or destroy existing competence and market links, and the honest position in an era of ferment is that you cannot tell which

quadrant you are in until you try. The stale-build field test taught me more in a day than the month of building did, and the most useful output was a write-up, not a score.

Be able to change. Innovation is inherently experimental, so it's hard to foresee the future, and thus we need to be able to change. The pieces are converging and they haven't converged. So I don't have a battle-tested concept to hand a customer, and I wouldn't claim today's shape is still the right one in a year.

Build versus buy resolves per customer, and for me it turns on whether there's one project I can maintain and share across several of them, instead of a bespoke build each time. Databricks and Snowflake are good products. Lifting the analysis out of them and tailoring it to a specific company still seems worth doing, and I lean open source because I'm in this for the long haul with these customers and want them able to migrate when they outgrow a choice I made for them. That preference colours most of what I end up recommending, so know that it is a preference. I lead the AI work at work, and all of the above is my opinion, not the company's position.

What I don't have

A regression suite over the semantic model. There's a `store_query` tool behind a `WREN_MCP_ALLOW_WRITE` flag that writes confirmed NL→SQL pairs back into `knowledge/sql/`, so this week's good answer becomes next week's recall hit, and that's the intended counter-measure to the model going stale as the warehouse moves under it. Whether it works, I can't yet say. Anthropic measured their own offline accuracy drifting from about 95% at launch to about 65% over a month before they treated it as an engineering problem. If I were putting this in front of a customer, that suite would go in before the first deployment, not after the first wrong number. I said as much in the presentation. I have not built it.

The next thing I want to try is Power BI as the semantic layer via Microsoft's remote MCP server, which executes DAX and returns the full model schema. A Copilot licence is only needed for the `Generate Query` tool, so if our agent writes the DAX the Fabric Copilot capacity requirement goes away and the customer's existing model becomes layer two. That is the dominant design absorbing the discontinuity, which is the other way an era of ferment can end, and it would be foolish to rule it out. It's still Preview, and row-level security is enforced for user auth but not for service-principal auth, so anything headless bypasses RLS entirely. That has to be designed around before it can be delivered, and I don't yet know how.

If you build one of these, two suggestions. Add a `server_info` tool before you run a field test, and write down what revenue means before anyone asks the model. The schema will happily tell you 1672.

References

- Utterback, J. M., & Abernathy, W. J. “A dynamic model of process and product innovation”. *Omega*, vol. 3, no. 6, pp. 639-656, 1975. doi:10.1016/0305-0483(75)90068-7
- Anderson, P., & Tushman, M. L. “Technological Discontinuities and Dominant Designs: A Cyclical Model of Technological Change”. *Administrative Science Quarterly*, vol. 35, no. 4, pp. 604, 1990. doi:10.2307/2393511
- Arthur, W. B. “Competing Technologies, Increasing Returns, and Lock-In by Historical Events”. *The Economic Journal*, vol. 99, no. 394, pp. 116, 1989. doi:10.2307/2234208
- Lindfors, E. T. “Radical path transformation of the Norwegian and Tasmanian salmon farming industries”. *Regional Studies, Regional Science*, vol. 9, no. 1, pp. 757-775, 2022. doi:10.1080/21681376.2022.2148555
- Abernathy, W. J., & Clark, K. B. “Innovation: Mapping the winds of creative destruction”. *Research Policy*, vol. 14, no. 1, pp. 3-22, 1985. doi:10.1016/0048-7333(85)90021-6