

AI Terms & Technology

A field guide to the language behind the WWG AI Dashboard project

Prepared for management orientation · June 2026 · Lockersoft

This guide explains, in everyday language, the AI terms that come up in our dashboard research — what they mean, why they matter for us, and how the pieces fit together. No technical background needed. Each term gives a short plain definition and, where useful, an everyday analogy. The first time an acronym appears, its full name is spelled out in parentheses. Six diagrams illustrate the big ideas, and the final section lays out what we recommend and roughly what it costs.

The one-sentence version of the project: we want our managers to ask plain-English questions and get answers drawn from our own company data — both the *structured* numbers in our SQL (Structured Query Language) database and the *free-form notes* our staff write — with that data kept inside a sealed, portable software package we control and encrypt with our own keys, whether it runs on our servers or on rented hardware.

The big picture: how a question becomes an answer

Before the vocabulary, here is the whole system on one page. When a manager types a question, it travels through these steps. The terms in **bold** are all defined later in this guide.

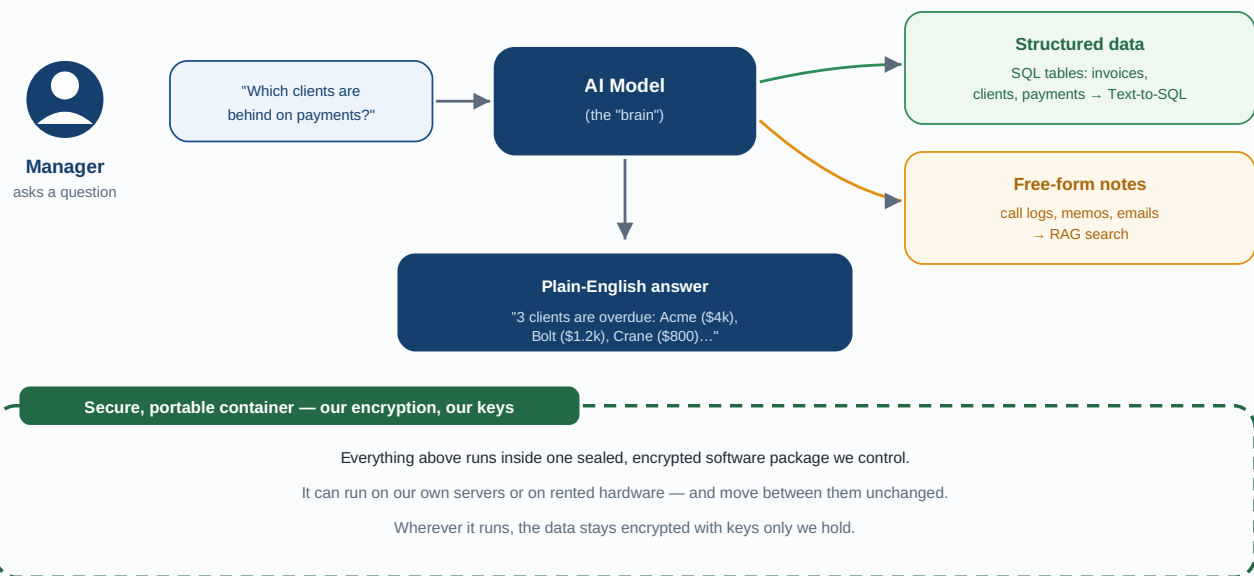


Figure 1. The end-to-end flow. A question goes to the AI model, which pulls supporting facts from two places — the SQL database (via *Text-to-SQL*) and the free-form notes (via *RAG*, Retrieval-Augmented Generation) — then writes a clear answer. The green dashed boundary is the security model: it all runs inside one sealed, portable container we encrypt and control, wherever we choose to host it.

Part 1 · The model itself

These terms describe the "brain" at the center of the system.

AI Model / LLM (*LLM = Large Language Model*)

A computer program trained to understand and produce human language. It can read a question and write a sensible answer. A Large Language Model (LLM) is the specific type that powers tools like ChatGPT — and the type we'd use here.

Analogy: a very well-read new employee who has read enormous amounts of text and can discuss almost anything — but who knows nothing about *our* company until we show them our data.

Parameters / Weights (*measured in "billions", e.g. 14B*)

The millions or billions of internal numbers a model adjusts as it learns. They are the model's "knowledge", baked into a set of files. A "14B" model has 14 *billion* of these numbers. More parameters generally means a smarter (but heavier and slower) model.

Analogy: the connections in a brain. The "weights" are how strongly each idea links to the next. "Model weights" and "the model files" mean essentially the same thing — the actual product you download and run.

Open-weight model (vs. *proprietary* / *closed*)

A model whose files are published for anyone to download, keep, and run on their own computers — for example Qwen, Llama, Phi, Gemma. A *proprietary* model (like the one behind ChatGPT) can only be used through someone else's website or service; you never possess it.

Why we care: with an open-weight model we own a permanent copy. Even if the company that made it disappeared tomorrow, our copy keeps working forever. This is central to our "what if a vendor goes under" concern.

Instruct model

A version of a model that has been trained to *follow instructions* and answer questions politely and helpfully, rather than just continuing text. The "-instruct" or "-chat" versions are the ones you actually talk to.

Analogy: the difference between a person who can read aloud and a person trained to actually *help you* when you ask for something.

Token

The small chunks of text a model reads and writes — roughly a word or part of a word. "Don't" is two tokens ("Don" + "'t"). Models measure their work, and vendors often charge, *per token*. A rough rule: 1,000 tokens $\approx$ 750 words.

Analogy: tokens are to an AI what syllables are to us — the bite-sized units it processes one at a time.

Context window

How much text the model can "hold in its head" at once — the question, plus any supporting data we feed it, plus its answer. Measured in tokens (e.g. 32,000 or 128,000). If we hand it too much at once, the oldest material falls out of view.

Analogy: the size of the desk it can spread papers on. A bigger desk lets it consider more documents at the same time.

Part 2 · Giving the model our knowledge

A fresh model knows the world in general but nothing about Lockersoft. There are three ways to fix that — from cheapest/easiest to most involved.

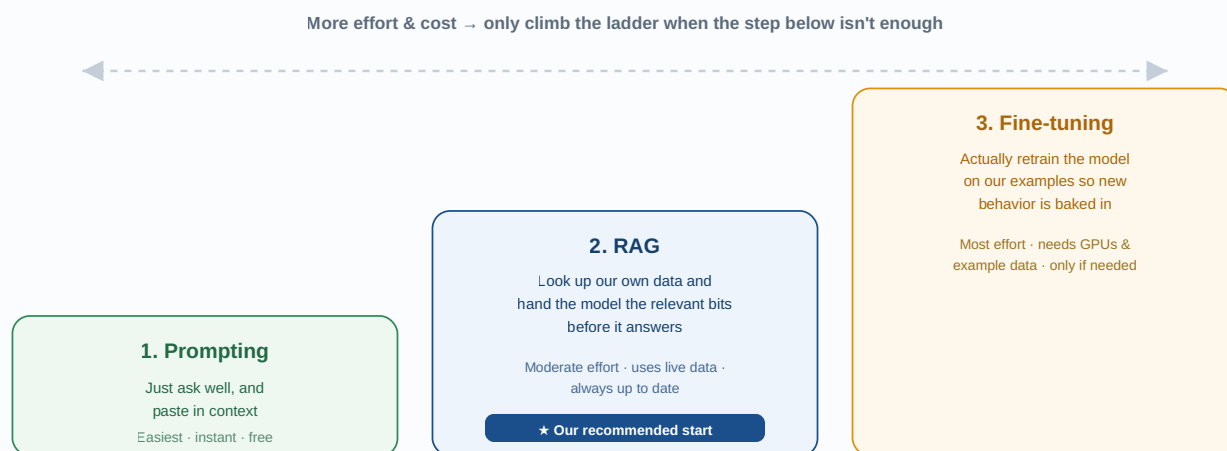


Figure 2. The three ways to give a model our knowledge, as an escalation ladder. Most of the value comes from steps 1 and 2. *Fine-tuning* (step 3) is powerful but is a last resort, not a starting point — a key finding from our research.

Training *(pre-training)*

The original, enormous process of building a model from scratch by feeding it a large share of the internet. This takes thousands of expensive computers and is done by the likes of Meta, Google, and Alibaba. **We would never do this ourselves** — we start from a model someone else has already trained.

Analogy: raising and educating a person from birth through university. We don't do that; we hire the already-educated graduate.

Fine-tuning *(also "tuning")*

Taking an already-trained model and giving it a focused extra course on *our* specific examples, so a new skill or style becomes permanent. It nudges the model's existing knowledge rather than rebuilding it. Useful when we need a consistent format or specialized jargon the base model keeps getting wrong.

Analogy: sending that experienced graduate on a one-week company training course so they learn our house style and terminology.

LoRA / QLoRA *(LoRA = Low-Rank Adaptation; QLoRA = Quantized Low-Rank Adaptation)*

A clever shortcut for fine-tuning. Instead of rewriting the whole model (huge, slow, expensive), LoRA bolts on a small "adapter" — a tiny add-on file holding just the new skill. QLoRA is the same idea made even lighter so it fits on a single modest GPU (Graphics Processing Unit).

Analogy: rather than reprinting an entire textbook to add a chapter, you slip in a thin booklet of new pages. The adapter file is small, portable, and ours to keep.

Adapter

The small file produced by LoRA fine-tuning that contains the new skill. It clips onto the base model. Because it's a standard, downloadable file, we can move it between systems freely — an important point for avoiding vendor lock-in.

RAG — the method we'll lean on most

RAG (*Retrieval-Augmented Generation*)

The technique of *looking up* relevant information from our own files and handing it to the model right when it answers — rather than hoping the model already memorized it. "Retrieval" = find the relevant notes; "Augmented Generation" = the model writes its answer using them.

Analogy: an open-book exam. The model doesn't need to have everything memorized; it's allowed to look up our documents and quote from them. This keeps answers current and grounded in real company data.

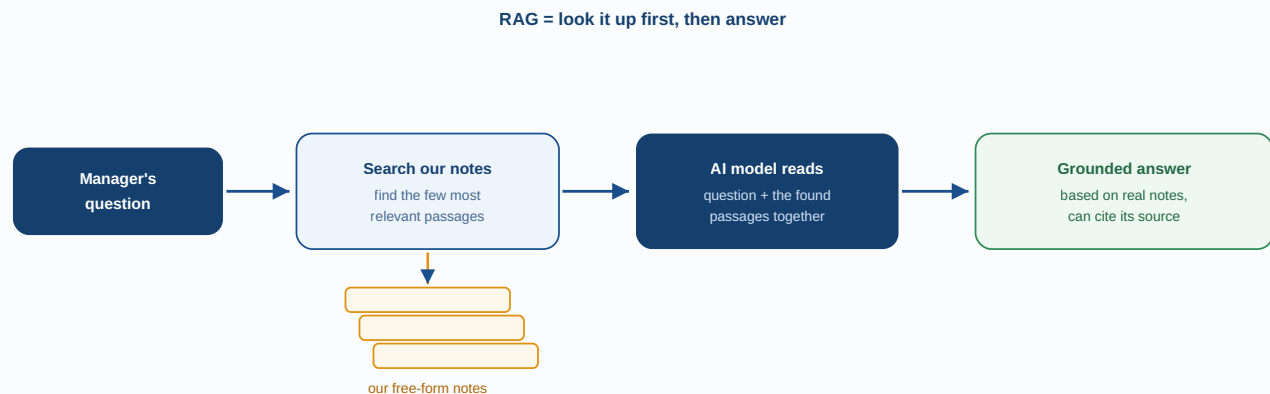


Figure 3. How RAG (Retrieval-Augmented Generation) answers a question about our notes. The system finds the handful of most relevant passages and gives them to the model alongside the question, so the answer is grounded in our actual records instead of guesswork.

Embedding

The trick that makes the "search our notes" step work. An embedding turns a piece of text into a list of numbers that captures its *meaning*. Two passages about "late payment" end up with similar numbers even if they use different words ("overdue invoice", "hasn't paid yet"). This lets the computer find things by meaning, not just exact keywords.

Analogy: giving every note a precise "GPS coordinate" of its meaning, so notes about similar topics sit near each other on a map.

Vector & Dimensions

The list of numbers an embedding produces is called a *vector*. The number of values in that list is its *dimensions* (e.g. 1,024). More dimensions can capture finer shades of meaning, but take more storage. (Practical note for us: the version of SQL Server we'd use can store vectors up to 1,998 dimensions, so we'll pick an embedding model that fits.)

Analogy: a location on a map needs 2 dimensions (north–south, east–west). Capturing the full meaning of a sentence needs hundreds — each dimension is one more nuance of meaning.

Text becomes numbers; similar meanings land close together

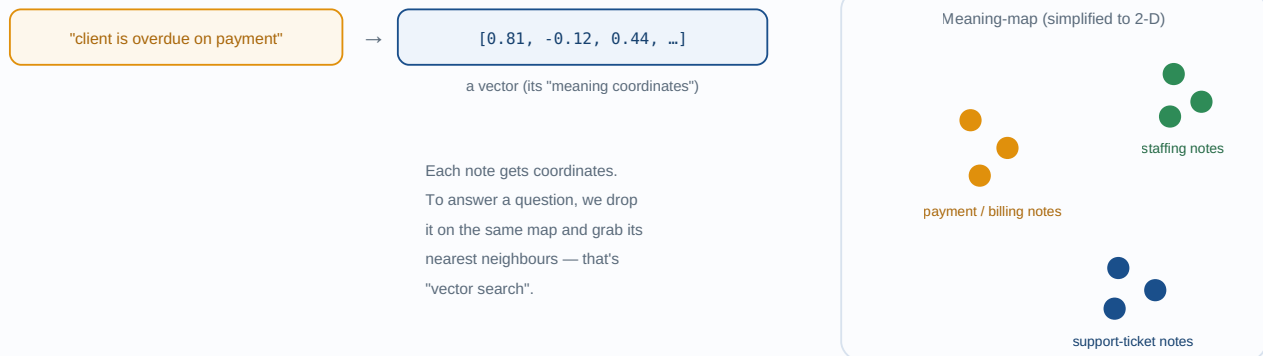


Figure 4. Embeddings in action. Text is converted to coordinates (a vector); notes about the same topic cluster together. Finding the nearest points to a question is how the system retrieves relevant notes — this is *vector search*.

Vector search / similarity

The act of finding the stored notes whose meaning-coordinates are closest to the question's. "Closeness" is measured mathematically (a common method is called *cosine similarity*). This is the engine under RAG's "find the relevant passages" step.

Vector database

A store built to hold these vectors and find nearest matches quickly. Good news for us: modern SQL Server can do this *inside the database we already use*, so the note-meanings live right next to our regular data — nothing has to be shipped to a separate system.

Chunking

Before we can embed notes, we cut long documents into bite-sized passages (a few paragraphs each). This keeps each piece focused so search returns a tight, relevant snippet rather than a whole 20-page file.

Analogy: indexing a book by paragraph instead of just by title — you find the exact relevant passage, not just "it's somewhere in this book".

Text-to-SQL

The counterpart to RAG, but for our *structured* data. The model translates a plain-English question ("total sales last quarter") into a database query (SQL) it can run to fetch exact numbers. We keep this safe by only allowing read-only look-ups and by showing the query for review.

Analogy: a bilingual translator who turns the manager's English request into the precise language the database speaks, then reads the result back in English.

Part 3 · Running it efficiently

These terms are about the hardware and the cost/speed trade-offs of actually operating a model.

Inference vs. Training

Two different activities, often confused. **Training** (and fine-tuning) is *building or teaching* the model — occasional, heavy, expensive. **Inference** is *using* the finished model to answer a question — constant, lighter, what happens every time a manager clicks "ask". Most of our day-to-day running cost is inference.

Analogy: training is writing and rehearsing the play; inference is performing it each night.

GPU & VRAM (GPU = Graphics Processing Unit; VRAM = Video Random-Access Memory)

The GPU is the specialized chip that runs AI models fast. Originally built for video-game graphics, GPUs turn out to be ideal for AI math. **VRAM** is the GPU's onboard memory; a model must fit inside it to run. The amount of VRAM you have decides how big a model you can run — this is the main hardware constraint and cost driver.

Analogy: the GPU is the engine; VRAM is the size of its fuel tank. A bigger model needs a bigger tank.

Quantization / "4-bit" (GGUF = GPT-Generated Unified Format; AWQ = Activation-aware Weight Quantization)

A compression technique that shrinks a model by storing its numbers with less precision — e.g. rounding to "4-bit" instead of the full "16-bit". This makes the model up to ~4× smaller and faster, so it fits on affordable hardware, with only a tiny quality loss (typically 1–3%). *GGUF* and *AWQ* are popular file formats for quantized models; *Q4_K_M* is a common, well-balanced 4-bit setting (the label simply means "4-bit, medium-quality variant").

Analogy: saving a photo as a high-quality JPEG — far smaller file, and you can barely tell the difference. "4-bit" is the sweet spot of small-but-still-sharp.

Quantization: same model, smaller footprint (example: a 14-billion-parameter model)



Figure 5. Why "4-bit" matters. Quantization lets a capable model run on hardware we can actually afford, with quality loss most users never notice. It's the difference between needing a \$30k server and a single mainstream GPU.

Model size "sweet spot"

Bigger isn't always better for us. Because RAG *supplies the facts*, the model doesn't need to have memorized everything. Our research points to a **12–14 billion-parameter model at 4-bit** as the sweet spot — smart enough for manager Q&A, small enough to run on a single affordable GPU with fast responses.

Part 4 · Where it runs & keeping data safe

These terms address our central requirement: data must stay secure — sealed in a container we encrypt and control — and stay portable, so we're never locked to one place.

Container / Docker

A way of packaging software so it runs identically anywhere — the whole application, its model, and all its dependencies sealed into one portable box. *Docker* is the most common tool for this. The container is the heart of our security-and-portability plan: we encrypt it, control it, and can lift the same box from our own servers to rented hardware and back.

Analogy: a shipping container — whatever's inside, it fits on any truck, ship, or train without repacking, and travels sealed and locked the whole way.

On-premises ("on-prem")

Running the software on hardware we own and control, inside our own building. It's the simplest way to guarantee data stays put — but, importantly, it's now *one option among several*, not a requirement. Because everything lives in a secure portable container, we can also run it on rented hardware with the same protection.

Analogy: cooking in your own kitchen. Great control — but if a sealed, tamper-proof lunchbox keeps the food just as safe, you're free to eat elsewhere too.

Cloud / Hyperscaler

Renting computing power over the internet from a big provider. "Hyperscaler" refers to the giants — Amazon (AWS, Amazon Web Services), Microsoft (Azure), and Google Cloud. Convenient and scalable; with the security measures below, we can use them while keeping our data sealed and our keys to ourselves.

Confidential Computing / TEE (*TEE = Trusted Execution Environment*)

The technology that lets us run on someone else's hardware while keeping data secret *even from that provider*. A TEE is a hardware-locked, encrypted "vault" inside the chip; data stays encrypted not just on disk and over the network but **even while it is being processed**. The system can also prove (via "attestation") that it's genuine before any keys are handed over. As of 2026 this is available on the AI-grade GPUs we'd use, with only a small speed cost.

Analogy: renting an office, but working inside a soundproof, windowless safe that only you can open — the landlord owns the building but can never see or hear what's inside.

Customer-Managed Keys (*CMK, also "BYOK" = Bring Your Own Key*)

The encryption keys that unlock our data are held and controlled *by us*, not the hosting provider. Combined with confidential computing, a key is only released after the system proves it's genuine — so even the provider's own staff can't read our data. This is what makes "data outside our walls" acceptably safe.

Analogy: you rent the safe, but you keep the only key — and it only turns once the safe confirms it hasn't been tampered with.

Data residency

The principle (and sometimes legal requirement) that data stays in a specific, controlled location or condition. Our standard is that data stays *sealed in our encrypted container under our keys*, wherever that container happens to run. Some cloud features can weaken this silently (e.g. "global" service tiers), so it's something to verify carefully.

VPC (*Virtual Private Cloud*)

A walled-off, private section of a cloud provider's data center reserved just for us. Better isolation than shared cloud. With confidential computing and our own keys layered on top, a VPC becomes a perfectly reasonable place to run our container.

Air-gapped

A system deliberately disconnected from the public internet, so data physically cannot leak out. The open-source tools we'd use *can* run fully air-gapped — the strongest possible posture — though with the secure-container approach we don't strictly require it.

Analogy: a vault with no doors to the outside. Whatever's inside simply cannot escape.

Vendor lock-in

Becoming so dependent on one supplier that leaving is painful or impossible. Our whole approach — open-weight models we own, standard containers, downloadable files, our own keys — is designed to *avoid* lock-in, so no single company's price hike, shutdown, or acquisition can strand us.

Part 5 · What we recommend & what it costs

Putting the vocabulary to work: here is the approach we're proposing and a realistic budget. Costs assume modest internal use — roughly 10–25 managers and a few hundred to ~2,000 questions a day. Figures are June 2026 estimates and will vary with usage and hardware prices.

The recommended approach in plain terms. Build the system as one *secure, portable container*: an open-weight 12–14B model (4-bit) that we own, answering questions via RAG over our notes and Text-to-SQL over our tables, with the note "meanings" stored right inside SQL Server. Encrypt everything and hold our own keys. Because it's a standard container, we can run it on our own server, on a rented secure GPU, or in a confidential-computing cloud — and move it between them without rebuilding. Start with **RAG only**; fine-tune later only if a real need appears.

Where we can run it (all the same container)

Option	What it is	Data protection
Our own server	A GPU box on-premises that we own.	Full — never leaves our building.
Rented secure GPU	A dedicated GPU from a vetted provider, billed by the hour or only when in use.	Strong — single-tenant, encrypted, our keys.
Confidential-computing cloud	A GPU in a hardware "vault" (TEE) where even the provider can't see the data.	Strongest off-site — cryptographically provable.

What each option costs per month

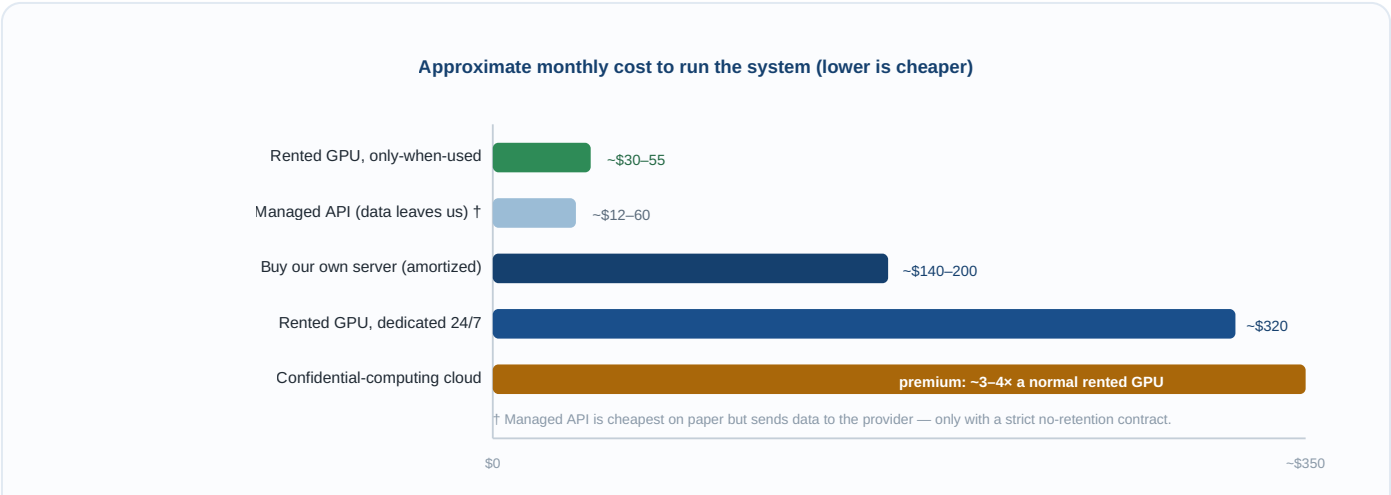


Figure 6. Rough monthly running costs. Supporting pieces (the embedding model and vector storage) are essentially free because we self-host them and keep the vectors inside SQL Server. A one-time fine-tune, if ever needed, is under ~\$100.

Cost item	Estimate	Notes
Buy a GPU server	~\$5,000–7,000 once	A 24 GB-class card (e.g. RTX 4090) handles this workload. ≈ \$140–200/mo spread over 3 years.
Rent, only-when-used	~\$30–55 / month	Cheapest defensible option for light, bursty use; data single-tenant.
Rent, dedicated 24/7	~\$320 / month	Always-on, no hardware to buy.
Confidential cloud	~3–4× a normal GPU	Only when we need provable "operator can't see it" security.
Embeddings + vector storage	~\$0	Self-hosted embedding model; vectors stored inside SQL Server we already run.
One-off fine-tuning	under ~\$100 (if ever)	A few hours on a rented GPU; we keep the resulting file. Usually unnecessary with RAG.

Buy vs. rent, in one line: buying a ~\$5–7k server pays for itself versus 24/7 rental in about **19 months**; over 3 years, owning costs roughly half. But **only-when-used rental (~\$30–55/mo)** can beat even owning for light usage. **Suggested path:** start on only-when-used rental to validate cheaply → redeploy the *same container* to confidential cloud if we ever need the strongest guarantee → buy a server if usage proves heavy and permanent. We can switch between all three without re-engineering — which is the entire point of the container-first design.

Quick-reference cheat sheet

Every term in one place, for skimming. Acronyms are shown with their full names.

Term	In one line
LLM (Large Language Model)	The language-understanding "brain" at the center of the system.
Parameters / weights	The billions of internal numbers that hold a model's knowledge; "14B" = 14 billion.
Open-weight model	A model we can download and own permanently (vs. one we only rent access to).
Instruct model	A model tuned to follow instructions and answer helpfully.
Token	A word-sized chunk of text; the unit models read, write, and bill by.
Context window	How much text a model can consider at once ("the size of its desk").
Training (pre-training)	Building a model from scratch — done by tech giants, not us.
Fine-tuning / tuning	Giving a finished model an extra course on our examples to bake in a skill.
LoRA / QLoRA (Low-Rank Adaptation)	An efficient fine-tuning shortcut that produces a small, portable add-on file.
Adapter	That small add-on file holding the new skill; clips onto the base model.
RAG (Retrieval-Augmented Generation)	Look up our own data and feed it to the model before it answers ("open-book exam").
Embedding	Turning text into numbers that capture its meaning, so we can search by meaning.
Vector / dimensions	The list of numbers (vector) and how many values it has (dimensions).
Vector search	Finding stored notes whose meaning is closest to the question.
Vector database	A store for vectors with fast nearest-match search; SQL Server can do this for us.
Chunking	Cutting long documents into focused passages before embedding them.
Text-to-SQL	Translating a plain-English question into a database query for exact numbers.
Inference vs. training	Using a model (constant, light) vs. building/teaching it (occasional, heavy).
GPU / VRAM (Graphics Processing Unit / Video RAM)	The chip that runs AI fast / its onboard memory that the model must fit into.
Quantization / 4-bit	Compressing a model to run smaller/faster with minimal quality loss (like a JPEG).
GGUF / AWQ (quantized-model formats)	Common file formats for compressed (quantized) models.
Container / Docker	A portable, sealed software package that runs identically anywhere.

On-premises	Running on hardware we own — one option, no longer the only one.
Cloud / hyperscaler	Renting compute from AWS (Amazon Web Services) / Azure / Google Cloud.
Confidential computing / TEE (Trusted Execution Environment)	Running on rented hardware while keeping data secret even from the provider.
Customer-Managed Keys (CMK / BYOK)	We hold the encryption keys, not the host — so no one else can read our data.
Data residency	Keeping data in a controlled location/condition (for us: sealed in our encrypted container).
VPC (Virtual Private Cloud)	A private walled-off section of a cloud, isolated for us.
Air-gapped	Fully disconnected from the internet; data physically cannot leak.
Vendor lock-in	Over-dependence on one supplier; our design deliberately avoids it.

The takeaway for management: the project combines an *open-weight AI model* we own, with *RAG* and *Text-to-SQL* to ground answers in our real data, all sealed in a *secure, portable container* encrypted with *our own keys*. We can run it on our own servers or on rented hardware — protected the same way either place — and move it freely. The result is a manager-friendly question box that respects our two non-negotiables: our data stays secure under our control, and we're never trapped with a single vendor. Likely running cost is modest: roughly **\$30–320/month** depending on how we host it.