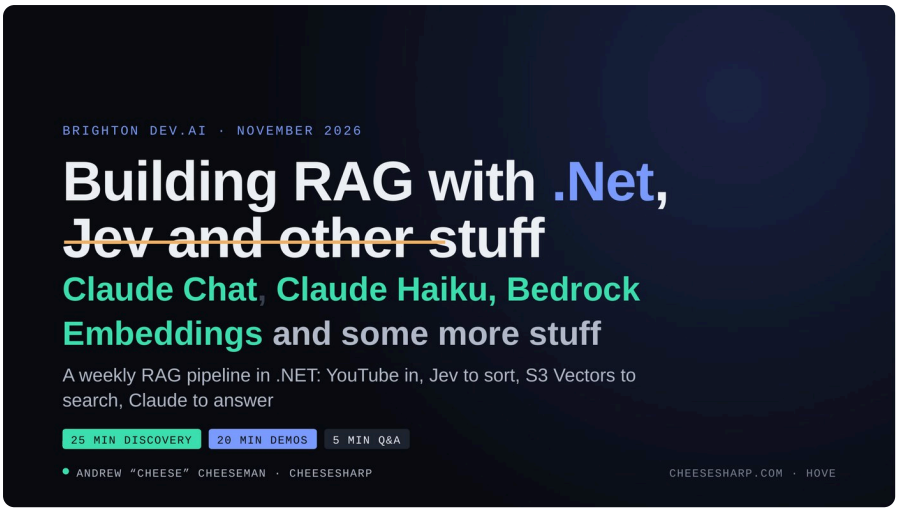



**1** / 37

## Building RAG with .Net, Jev and other stuff

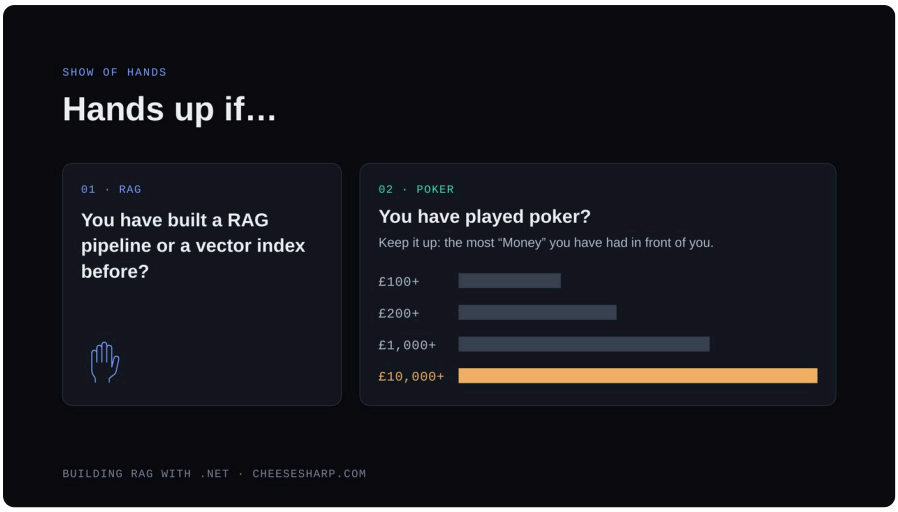
82 words · about 0:30 to read aloud

---

Name. Thirty years in infrastructure, twenty-five in C#, still writing code every week. Some of you were here in September for "From Zero to AI".

Tonight in three parts: what I am building, why it is about poker, and how I got here. Then how each piece works, then twenty minutes of demos on the real data, then questions. Everything you see is .NET. Shout if a demo breaks.

*[Confirm the date and venue with Gavin and update the eyebrow if needed.]*




**2** / 37

SHOW OF HANDS

## Hands up if...

0 words

---

*No speaker notes for this slide.*




# 3 / 37

## Picture slide

0 words

*No speaker notes for this slide.*

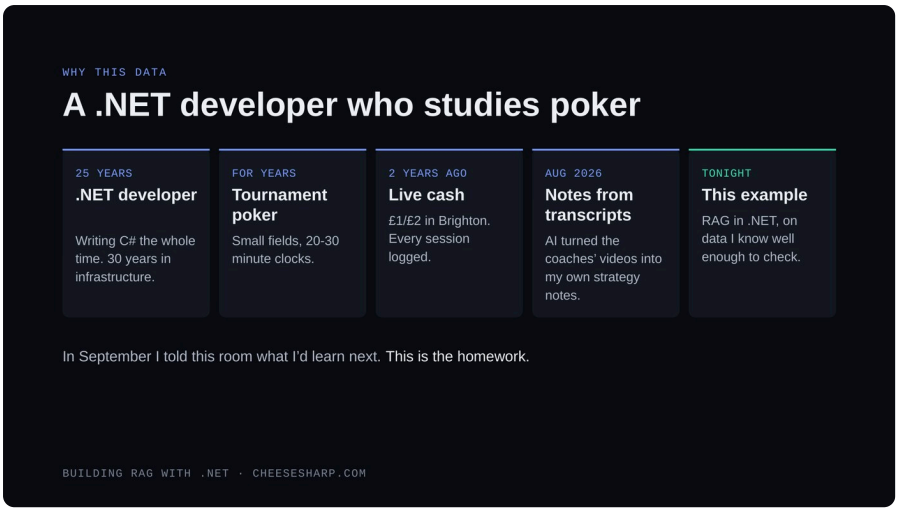



# 4 / 37

## Picture slide

0 words

*No speaker notes for this slide.*




**5** / 37

WHY THIS DATA

## A .NET developer who studies poker

189 words · about 1:15 to read aloud

---

Why this data? Because of who I am.

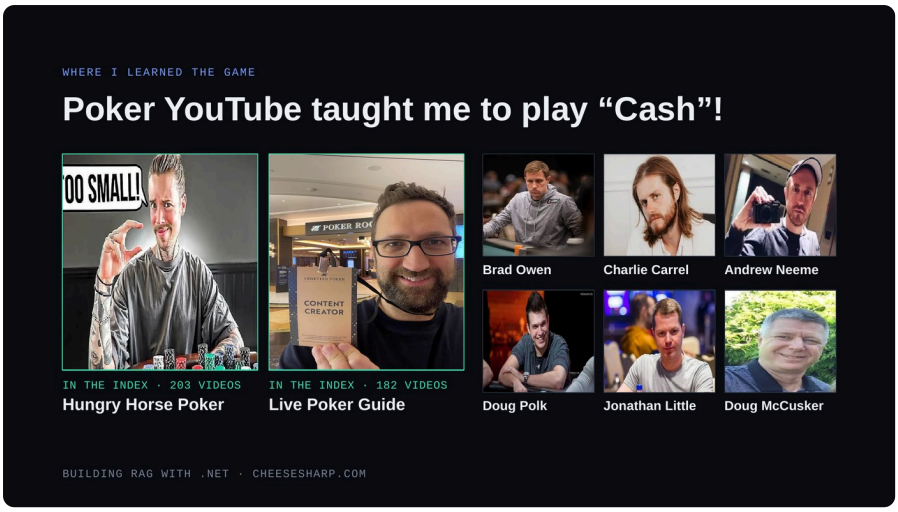
I am a .NET developer first. Twenty-five years of C#, thirty in infrastructure. That is why this talk is .NET from end to end: it is what I would build with on a Monday morning.

I also play poker, seriously enough to study it. Tournaments for years: small fields, 20-minute clocks. About two years ago I took up live cash, mostly £1/£2 here in Brighton, and I have logged every session since.

I study these two coaches closely. In August I used AI to turn their transcripts into my own strategy notes. That worked, up to a point. Next slide shows the point.

So tonight is a worked example. It is not a product. I picked this data because I am interested in it, and because I know it well enough to tell a right answer from a made-up one. That matters more than it sounds when you are judging a RAG system.

And it closes a loop with this room: in September I said what I would learn next, putting AI inside the software we build. This is the homework.



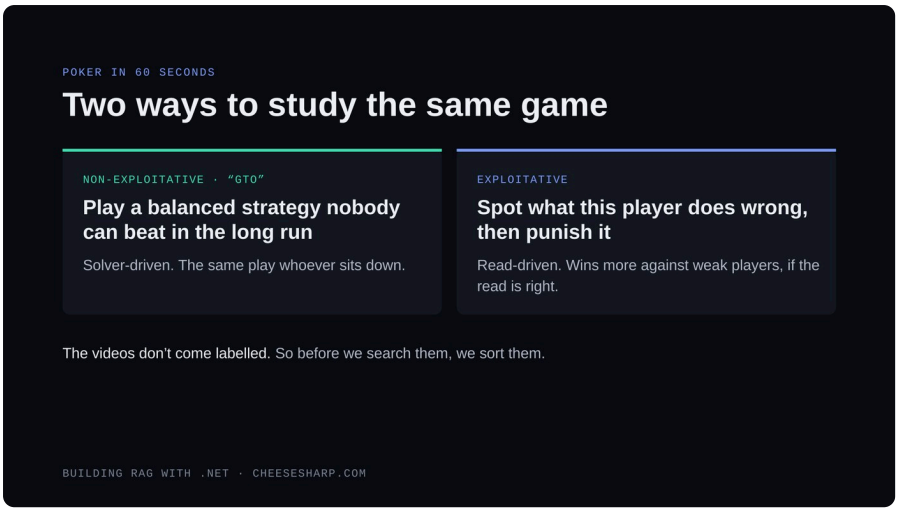

6 / 37

WHERE I LEARNED THE GAME

## Poker YouTube taught me to play “Cash”!

0 words

*No speaker notes for this slide.*




**7** / 37

POKER IN 60 SECONDS

## Two ways to study the same game

147 words · about 1:00 to read aloud

---

For anyone who does not play: there are two broad ways to study.

Non-exploitative, usually called GTO, game theory optimal: you learn a balanced strategy from solvers that cannot be beaten in the long run. It is the same play whoever is at the table.

Exploitative: you work out what this particular player does wrong (calls too much, never bluffs, folds too often) and you deviate to punish it.

Most live low-stakes coaching leans exploitative. [Optional, your own angle: you play live £1/£2 and study both; exploitative is how you play, equilibrium is the foundation underneath.]

The point for tonight: if I ask "how do I exploit a calling station?", I do not want a GTO lecture back, and I definitely do not want a travel vlog. The videos do not come labelled. So before we search anything, we sort it. That is where Jev comes in.




8 / 37

WHY POKER

## I’ve watched most of this. I can’t remember which video said what.

160 words · about 1:15 to read aloud

---

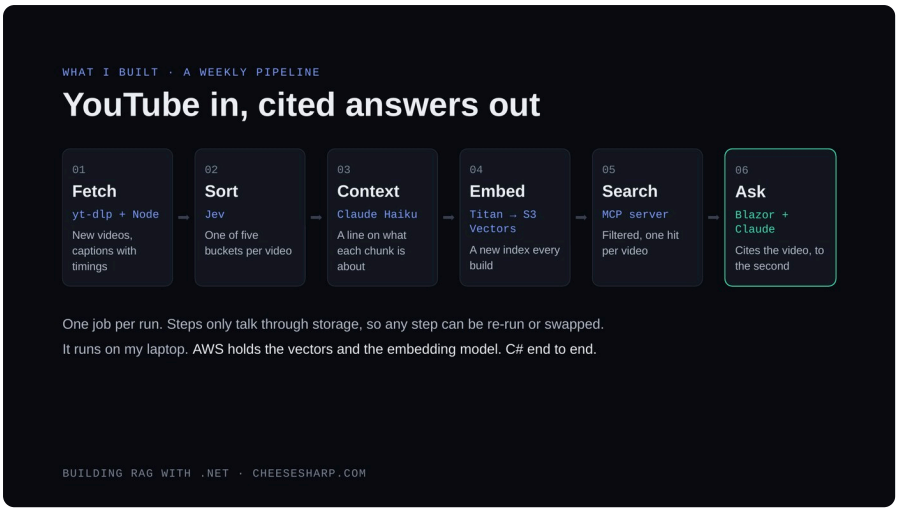
Why poker? Because the problem is small and personal, which makes it a good one to learn on. And as RAG data, poker is a gift: the captions mangle the jargon, a chunk of one channel is session vlogs, and the two coaches do not always agree.

I study poker from two YouTube coaches: Hungry Horse Poker and Live Poker Guide. Between them, 385 videos with captions, cut into 14,254 chunks.

I know I have heard the answer to "how do I play against someone who never bluffs the river" at least three times. I cannot tell you which video, which coach, or whether they agreed.

That is the shape of most real RAG problems: a pile of unstructured content you already own, and questions you cannot answer with Ctrl+F.

The numbers are exact, from the run report of the live index. Credit both channels by name. This is their work; I am only indexing my own copy for study.




**9** / 37

WHAT I BUILT · A WEEKLY PIPELINE

## YouTube in, cited answers out

265 words · about 2:00 to read aloud

---

Start with what I built. Then why poker, and how I got here.

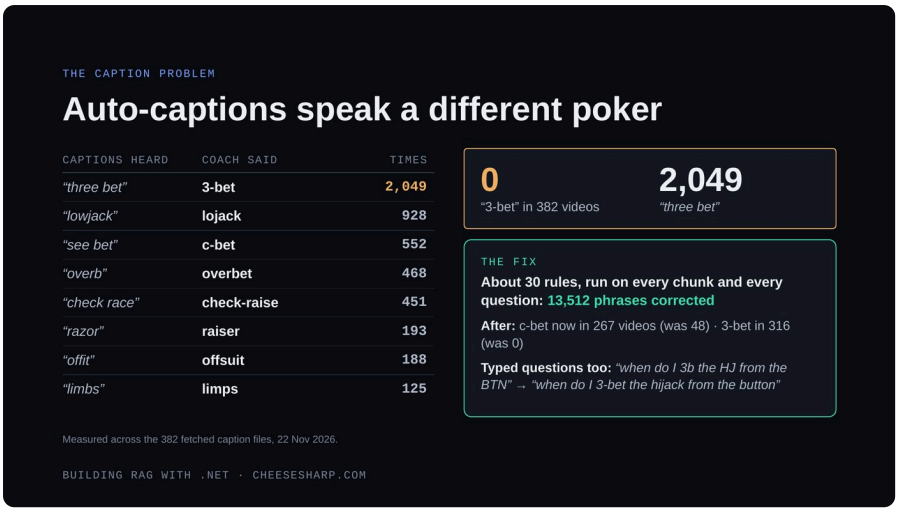
It is a weekly pipeline: YouTube goes in one end, and answers you can check come out the other. Six steps, one job each.

1. Fetch: pull any new videos from two coaching channels. yt-dlp does the fetching, .NET drives it, and it keeps the caption timings.
2. Sort: Jev, a new decision model from TypeSafe AI, puts each video in one of five buckets.
3. Context: Claude Haiku writes one line per chunk saying what it is about. Whether that helped is one of the best stories tonight.
4. Embed: Amazon Titan turns each chunk into a vector, and every build writes a fresh index into Amazon S3 Vectors.
5. Search: a small .NET MCP server. Filtered search, one hit per video, a link to the second.
6. Ask: a Blazor app where Claude coaches you, RAG on or off, citing the videos.

Each step is one job, run on its own, and steps only talk through storage, so any step can be re-run or swapped without touching the others.

It all runs on my laptop, with the data in a local storage folder. AWS holds the vectors, the embedding model and one pointer to the live index, which costs pennies a month. Claude and Jev are called over their own APIs. The fetch step has to run here anyway: YouTube rate-limits cloud IP ranges.

C# from one end to the other. Most RAG tutorials assume Python. I wanted to know if .NET is a first-class citizen now. Spoiler: it is.



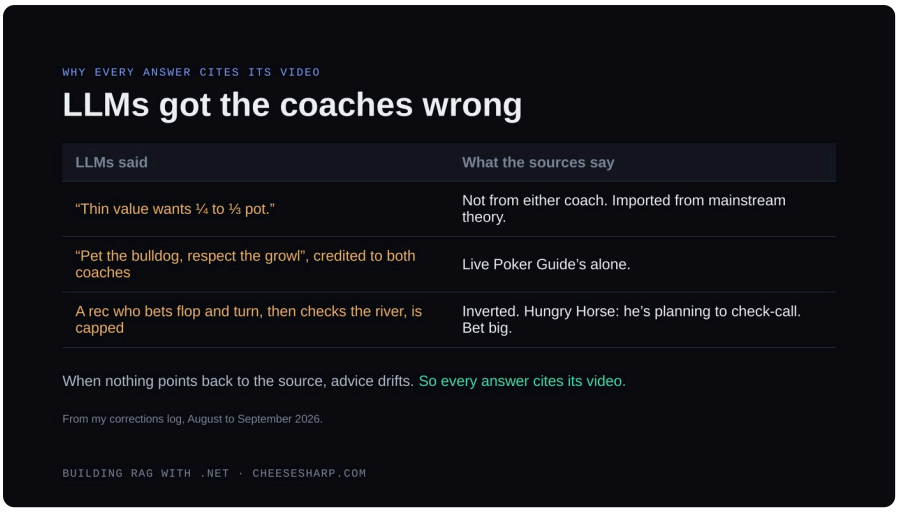

# 10 / 37

THE CAPTION PROBLEM

## Auto-captions speak a different poker

0 words

*No speaker notes for this slide.*




**11** / 37

WHY EVERY ANSWER CITES ITS VIDEO

## LLMs got the coaches wrong

241 words · about 1:45 to read aloud

---

This is the slide that turned a study habit into a requirement.

I keep a corrections log. A record that quietly edits its own errors is worth nothing, so every time a rule in my notes turned out to be wrong, it goes in, with what the source actually says.

Three of them, all from August and September:

"Thin value wants a quarter to a third of the pot." It sounded right. Neither coach says it. It came in from mainstream theory. Both coaches actually want much bigger value bets.

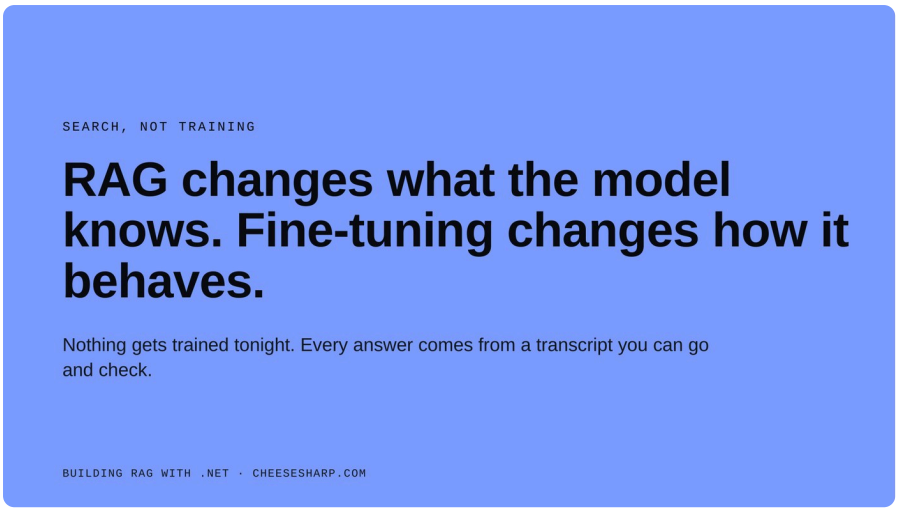
"Pet the bulldog, respect the growl" was credited to both coaches. It is Live Poker Guide's alone. Hungry Horse's first principle is "be water".

And the expensive one: a recreational player who bets flop and turn, then checks the river, is capped, so bet thin. Hungry Horse says the opposite: he is checking because he plans to check-call, and he only double-barrels his strongest hands. Bet big. That one cost me at the table.

Some of these came from me, some from the AI that helped write the notes. It does not matter which. The problem is the same: nothing pointed back to a source, so nothing caught them.

So the rule for this system: every answer cites the video it came from, and ideally the minute. If I cannot check it, I do not trust it.

Sources in the log: C1 (Hungry Horse, bx5YQWVtLSs), C2, C4 (Live Poker Guide, FjDsnLhFUjA).




# 12 / 37

## RAG changes what the model knows. Fine-tuning changes how it behaves.

130 words · about 1:00 to read aloud

---

This is the one sentence I would want you to remember if you forget everything else.

When people say "we need to train it on our data", nine times out of ten they mean "we need it to know our data". That is retrieval, not training.

Retrieval augmented generation: at question time, search your own content, hand the relevant bits to the model, and tell it to answer only from those. The model's weights never change.

Fine-tuning changes behaviour: format, tone, a narrow skill. It is slower, needs labelled examples, and it does not give you citations.

The bonus of retrieval: every answer can point at its source. For a study tool, that is the feature. I want to go and watch the bit of the video it came from.




**13** / 37

EMBEDDINGS AND VECTOR DATABASES

## Search by meaning, not by words

274 words · about 2:00 to read aloud

---

Four words you will hear all night. Two on this slide, two on the next.

Embedding. A model reads a passage and hands back a vector: 1,024 numbers for Titan Text Embeddings V2. Think of it as an arrow from the centre of a space with 1,024 dimensions. You cannot read the numbers, but you can compare the arrows.

Here are three of those dimensions. Titan normalises its vectors, so every arrow has the same length and every chunk is a dot on the surface of a sphere. Only the direction carries meaning.

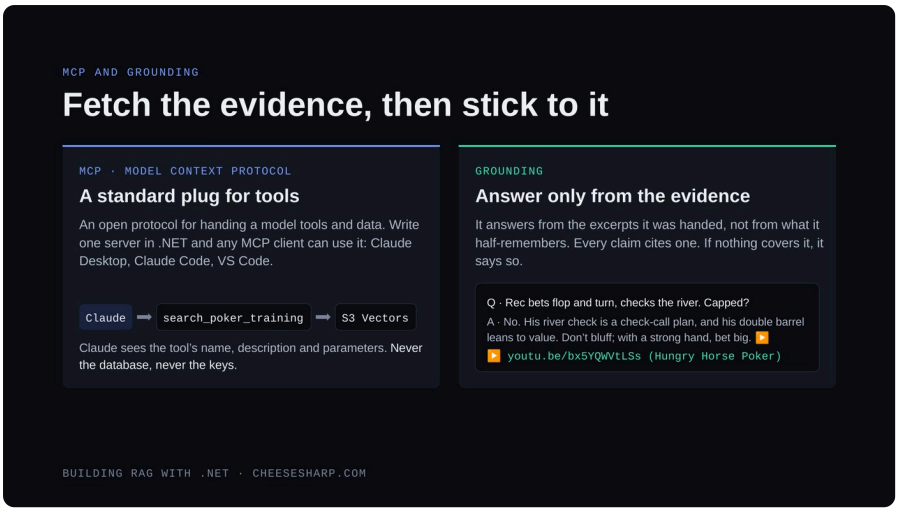
My question is the blue arrow. The chunks where the coaches talk about river bluffs sit right around its tip. One of them is about firing the third barrel when a scare card lands: not one word in common with my question, same direction. Keyword search misses it. Embedding search finds it. Preflop ranges are further round. The Vegas vlogs are off at a wide angle.

That angle is the measure. Cosine similarity is the cosine of the angle between two arrows: 1 means the same direction, near 0 means unrelated.

Vector database. It stores every chunk's vector with its metadata: channel, video, bucket. Give it your question's vector and it returns the closest in angle. Add a filter and it only looks where you told it to. Mine is Amazon S3 Vectors, with the index set to cosine distance.

In .NET, embedding is one call: GenerateVectorAsync on IEmbeddingGenerator, from Microsoft.Extensions.AI. Swap the model, the code stays the same.

*[The picture is an illustration, not a projection of my data. The live index: Titan Text Embeddings V2, 1,024 dimensions, cosine distance.]*




14 / 37

MCP AND GROUNDING

## Fetch the evidence, then stick to it

304 words · about 2:15 to read aloud

---

The other two words.

MCP, the Model Context Protocol. An open standard for giving a model tools and data. People call it USB-C for AI tools: I write one server in .NET, and any client that speaks MCP can use it: Claude Desktop, Claude Code, VS Code. Claude sees the tool's name, its description and its parameters. It never sees the database, and it never sees my AWS keys.

Grounding. A grounded answer comes from evidence handed to the model for this question, not from whatever it soaked up in training. Two habits make it real: every claim cites its source, and when the evidence does not cover the question, the answer says so.

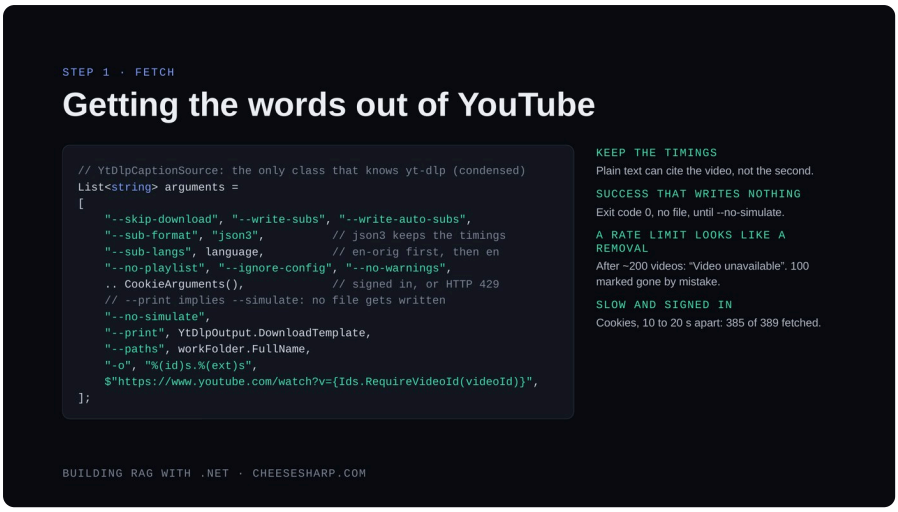
Here is the mistake from my corrections log again, answered grounded. My notes said a rec who checks the river is capped, so bet thin. The grounded answer says what the video says: his check is a check-call plan and his double barrel leans to value, so don't bluff, and bet big with a strong hand. And it links the video. With that link, I would have caught it in August.

So: RAG is the pattern. Embeddings and the vector database do the retrieval. MCP is how Claude reaches it. Grounding is the rule for what Claude does with what comes back.

Grounding cuts down made-up answers. It does not remove them. That is why the citation matters: you can check.

Hold on to this definition. Building it, it turned out to be a dial, not a switch: that's slide 22.

*[The answer on the slide paraphrases correction C1's source, video bx5YQWVtLSs, chunks 2 to 4. The app cites as [▶ mm:ss] (link to the second): ask this in demo 4 and copy the real second onto the slide. The context line on the chunking slide comes from the same video.]*




**15** / 37

STEP 1 · FETCH

## Getting the words out of YouTube

270 words · about 2:00 to read aloud

---

Step one is boring, and it is where a lot of the work went.

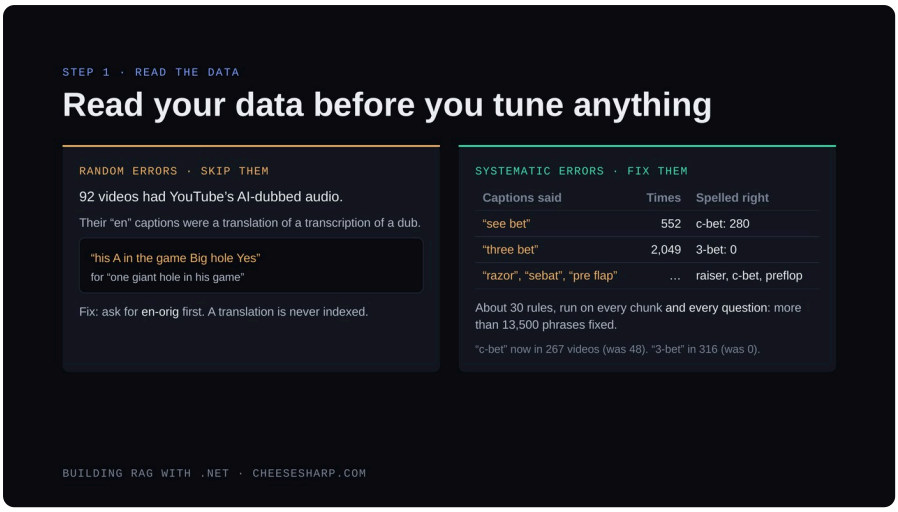
YouTube does not want you pulling captions programmatically, and it changes how it blocks you. yt-dlp is the tool that keeps up, so it does the fetching and .NET drives it. One class, YtDlpCaptionSource, knows yt-dlp; everything downstream sees plain records. When YouTube breaks things again, it is a one-class change.

What bit me:

- Keep the timings. My first corpus was plain text, and plain text can cite a video but not the second. json3 keeps every caption's start time.
- yt-dlp's --print quietly turns on --simulate. The command succeeds, exit code zero, and writes nothing. --no-simulate fixes it; the comment is in the code so nobody "tidies" it away.
- Rate limits look like removals. After about 200 videos at 3 seconds apart, YouTube answered "Video unavailable. This content isn't available, try again later". My code read that as a removed video and marked 100 videos as gone for good. Now rate-limit wording is checked first, the run stops at the first rate limit, and the manifest is saved after every video so it carries on where it stopped.
- Slow and signed in. Without a signed-in cookies file YouTube answered 429. With cookies and 10 to 20 seconds between videos: 385 of 389 fetched. The other four are age-restricted.

From the first version: the obvious Node library, youtubei.js, fetched zero transcripts in twelve tries. Measure before you commit to a dependency.

That is also why fetch runs on my laptop, not in AWS: the cookies live here, and YouTube rate-limits cloud IP ranges.




16 / 37

STEP 1 · READ THE DATA

## Read your data before you tune anything

249 words · about 1:45 to read aloud

---

Once you have the text, read it. Properly. Before you tune anything.

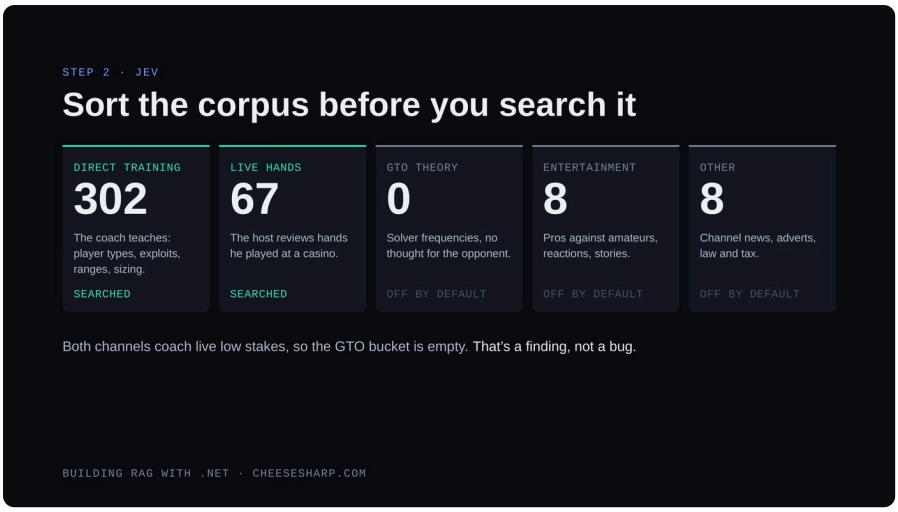
Two problems turned up, and they need opposite fixes.

Random errors: skip them. 92 videos, mostly Hungry Horse, carry YouTube's auto-dubbed audio in about sixteen languages. yt-dlp reported the first dub, Bengali, as the video's language, so the "en" captions were a machine translation of a transcription of a dub. Word salad: "his A in the game Big hole Yes" for "one giant hole in his game". There is no pattern to fix, so: ask for en-orig first, the real English transcription, and never index a translation. All 385 videos now come from en-orig.

Systematic errors: fix them. Speech recognition mishears poker jargon, and it mishears it the same way every time. "See bet" appeared 552 times; "c-bet", spelled right, only 280. "3-bet" never appeared at all; "three bet" appeared 2,049 times. That splits one idea across several spellings for the embeddings, and hides it completely from keyword search.

Because the errors are systematic, a short list fixes them: about 30 rules, mined from the captions and checked by hand. They run on every chunk before embedding and on every question before search, so both sides use one spelling. More than 13,500 phrases corrected. "c-bet" now appears in 267 videos, up from 48; "3-bet" in 316, up from zero. Typed shorthand too: "3b", "OOP", "HJ" are written out the way coaches say them.

Cheap beats clever: a regex list, deterministic and free, before any clever retrieval trick.






STEP 2 · JEV

## Sort the corpus before you search it

213 words · about 1:30 to read aloud

---

Step two is the one most RAG tutorials skip: sort before you search.

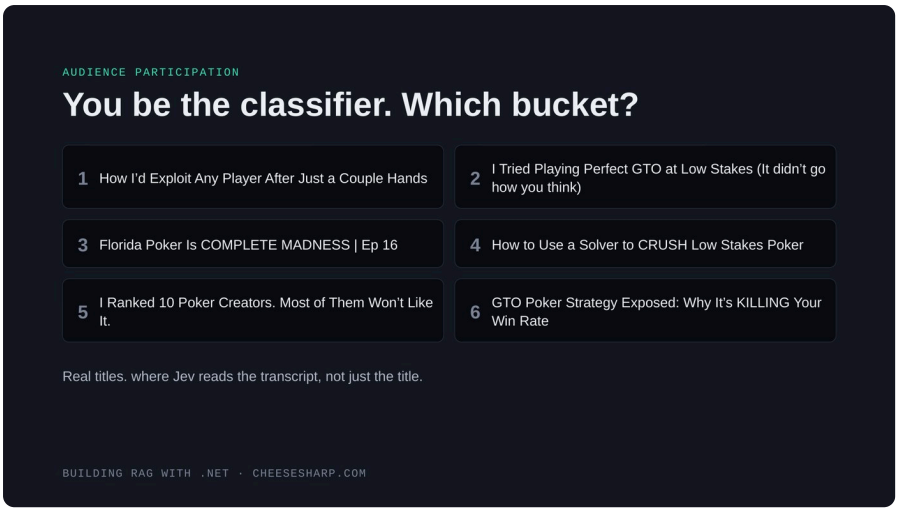
Every video goes into one of five buckets. The split is by whether the advice depends on the opponent, then by format:

- Direct training: the coach teaches, to camera or to a student. 302 videos.
- Live hands: the host reviews hands he played himself at a casino. Session vlogs, money challenges, trips. 67.
- GTO theory: balance and solver frequencies, no thought for the opponent. Zero.
- Entertainment: pros against amateurs, reactions, stories. 8.
- Other: channel news, adverts, law and tax. 8.

It took three goes. My first draft had four buckets and put fundamentals under non-exploitative. Wrong: fundamentals taught for real low-stakes games are exploitative training. The second run left one video in GTO and 307 in exploitative, so I split exploitative by format: live hands versus direct teaching.

The GTO bucket ended up empty. Its one video, a GTO Wizard tutorial, moved to direct training when classify read the real captions. Both channels coach live low stakes, so that is an honest result about the corpus, not a bug.

The app searches the two exploitative buckets by default. Ask how to beat a calling station, and you don't get a pros-against-amateurs show or channel news.




**18** / 37

AUDIENCE PARTICIPATION

## You be the classifier. Which bucket?

127 words · about 1:00 to read aloud

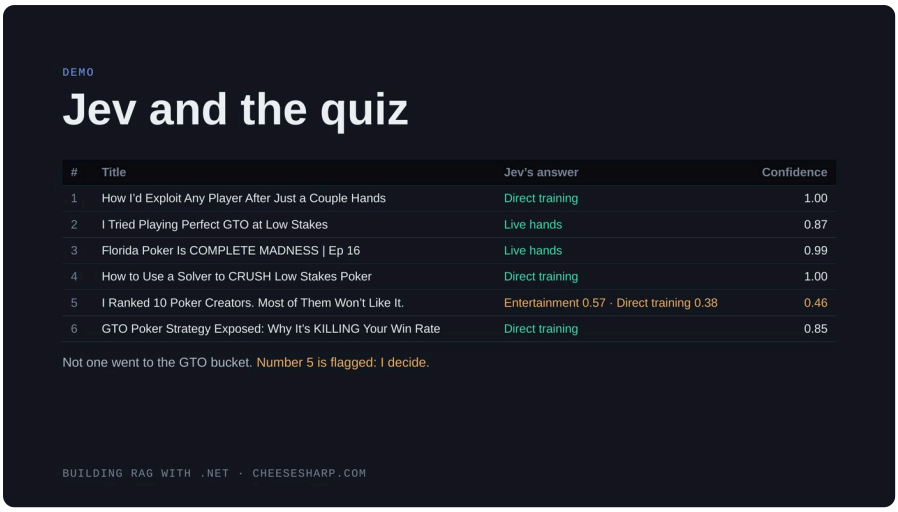
---

Your turn. Six real titles from the Hungry Horse channel. Shout a bucket: live hands, direct training, GTO theory, entertainment, other.

Let the room argue about 2, 4 and 6: two have "GTO" in the title and one has "solver". The title alone doesn't tell you whether the video teaches balanced play, teaches how to exploit people, or is the host's own session. Number 3 is easy: a numbered vlog episode. Number 5 looks like "other".

The point: titles are written to get clicks, not to be classified. That's why Jev reads the captions, not just the title. Answers in demo 2.

Video IDs: 1 BlPhIOyM_qk · 2 Omew9wgrmQA · 3 8RTta81K9qo · 4 REMDdZL01Ko · 5 vsSFecrDrb0 · 6 6urFJ_QJOdE. All six are in the index.




19 / 37

DEMO

## Jev and the quiz

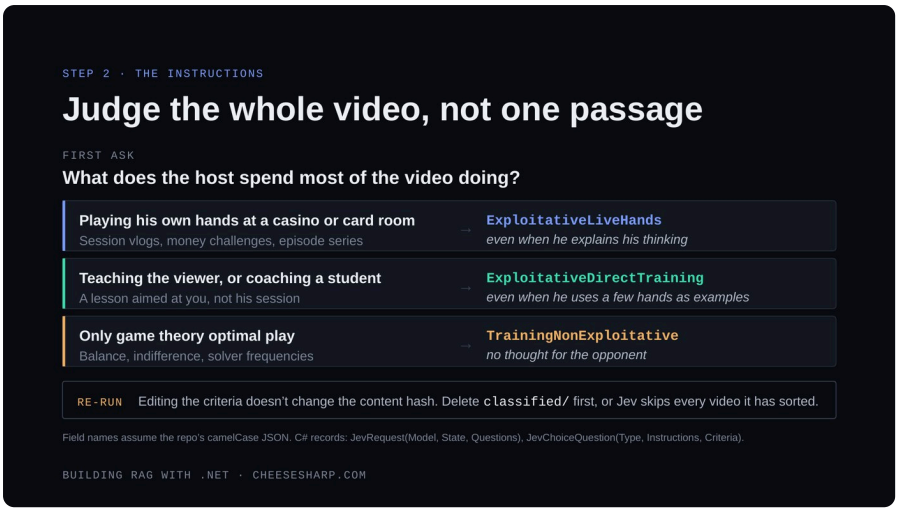
259 words · about 1:45 to read aloud

---

Goal: sorting as a real, checkable step, and the quiz payoff.

1. (1:30) Reveal the answers against what the room shouted. These are Jev's real answers from the 21 November run, on the slide in case the live call fails.
   - 2 has "GTO" in the title, but it's his own session at a casino: live hands.
   - 4 has "solver" in the title, but it teaches how to beat low-stakes players: direct training.
   - 6 is "GTO Strategy Exposed", and it's an argument for exploiting: direct training.
   - Not one of the six went to GTO theory, and neither did anything else in the corpus.
2. (1:30) The hard one: 5, "I Ranked 10 Poker Creators". Entertainment at 0.57, direct training at 0.38. Confidence 0.46, under the 0.7 threshold, so it's flagged for review. Show its probabilities. Who makes the call? Me. A review in storage wins over Jev's answer at the next embed.
3. (1:00) Run classify live on one transcript if there's a new video from demo 1, and show the typed result: bucket, confidence, probabilities, input tokens.
4. (1:00) Counts for the whole corpus from the run report: Direct training 302, Live hands 67, Entertainment 8, Other 8, GTO 0. Split by channel: Hungry Horse 141 / 53 / 7 / 2, Live Poker Guide 161 / 14 / 1 / 6.

Video IDs: 1 BlPhIOyM_qk · 2 Omew9wgrmQA · 3 8RTta81K9qo · 4 REMDdZL01Ko · 5 vsSFecrDrb0 · 6 6urFJ_QJOdE. Number 5 is one of the longest transcripts; classify keeps its first 100,000 characters.




# 20 / 37

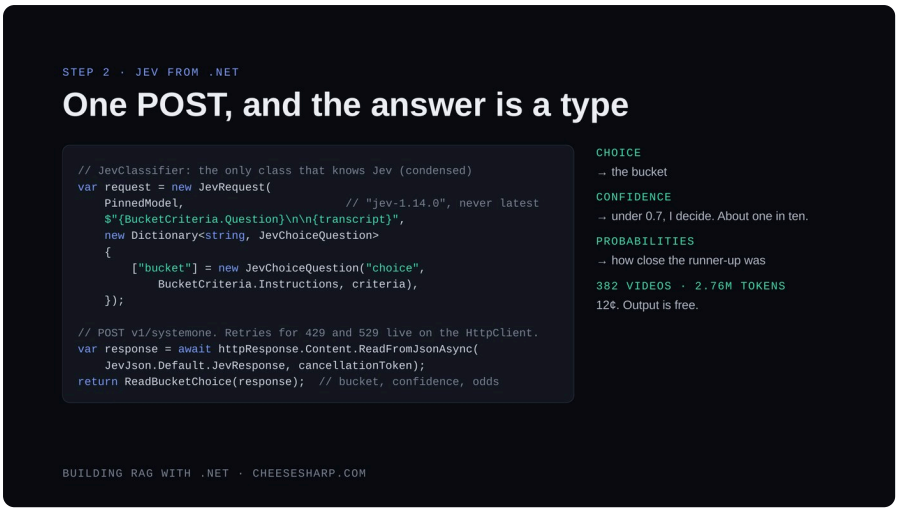
STEP 2 · THE INSTRUCTIONS

## Judge the whole video, not one passage

0 words

---

*No speaker notes for this slide.*






STEP 2 · JEV FROM .NET

## One POST, and the answer is a type

211 words · about 1:30 to read aloud

---

So what is Jev? The first model from TypeSafe AI. They call it a "System One" model: fast decisions, not slow reasoning. The important thing: it does not write. It decides.

You give it state, here a transcript, and typed questions. Three kinds: choice (pick one of my options), score (place it on a rubric I write), and a yes-or-no probability. I use one choice question per video: which bucket? The bucket descriptions are the criteria.

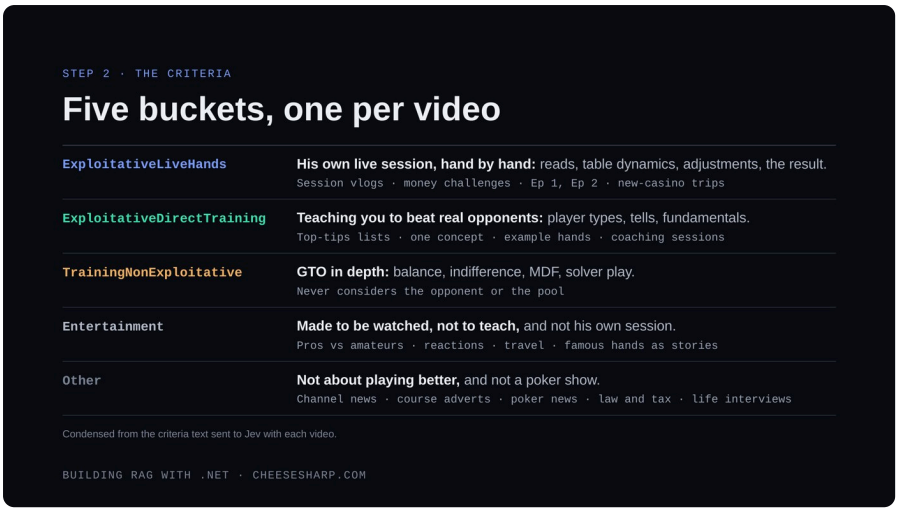
What comes back is a type, not a sentence: the choice, a confidence, and the probability of every option. No prose to parse.

From .NET it is plain HTTP. There's no official .NET SDK, the community ones are unverified, and one small class with retries on the HttpClient is clearer on a slide anyway. Microsoft.Extensions.Http.Resilience retries 429 and 529 with backoff and honours Retry-After. The model is pinned, jev-1.14.0, never jev-latest, so a re-run means the same thing.

Jev reads the start of each transcript: the first 100,000 characters, about 25,000 tokens, inside its 32K-per-question limit. The start of a video says what kind of video it is.

Anything under 0.7 confidence is flagged for me to review. About one video in ten.

Cost: 382 videos, 2.76 million input tokens, 12 cents. Output is free.




**22** / 37

STEP 2 · THE CRITERIA

## Five buckets, one per video

Hidden in the deck · backup slide

521 words · about 3:45 to read aloud

### INSTRUCTIONS

Pick the bucket that describes most of the video, not one passage of it. First ask what the host spends most of the video doing. If he is going through hands he played himself at a casino or card room, such as a session vlog, a money challenge or an episode of a series, choose ExploitativeLiveHands, even when he explains his thinking. If he is teaching a lesson directly to the viewer or to a student, choose ExploitativeDirectTraining, even when he uses a few hands as examples. Choose TrainingNonExploitative only when the whole lesson is about game theory optimal play: balance, indifference and solver frequencies, with no thought for the opponent.

CRITERIA (THE FIVE BUCKETS)

ExploitativeLiveHands: Most of the video is the host reviewing hands he played himself in a live game at a casino or card room. Typical formats: session vlogs, money challenges such as making $100 an hour or $10,000 a month, episode series (Ep 1, Ep 2 and so on), and trips to play in a new city or casino. He talks through each hand as it happened: what he put opponents on, how the table plays, the adjustments he made against them, and the result of the session.

ExploitativeDirectTraining: The host teaches the viewer directly, or coaches a student, rather than walking through his own session. The advice is for winning against the players you actually meet, mostly at live low stakes: player types and population tendencies, live tells, reads after a showdown, exploiting recreational players, calling stations and nits, punishing players who bluff too much or too little, and deviating from GTO on purpose. Includes fundamentals taught for these games: preflop ranges and open sizes, bet sizing for value against weak players, pot odds and other maths, hand reading, study methods and tools, bankroll management, mindset, tilt and dealing with downswings. Typical formats: top tips and mistakes lists, explaining one concept, example hands that illustrate a lesson, coaching sessions.

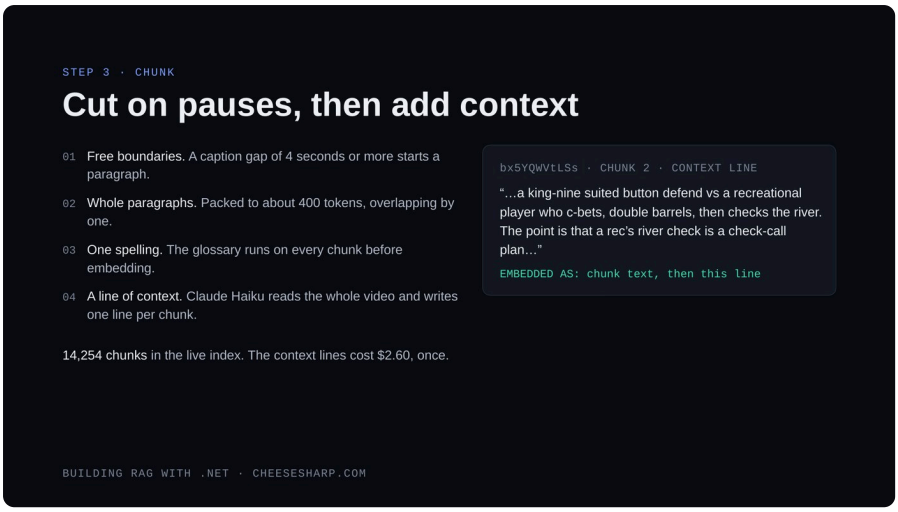
TrainingNonExploitative: Teaches game theory optimal (GTO) poker in depth: a balanced strategy that cannot be exploited, whoever the opponent is. The whole video stays with equilibrium ideas: balanced ranges and frequencies, making the opponent indifferent, minimum defence frequency, bluff to value ratios, mixed strategies, range and nut advantage, and what the solver does and why. It never considers how a particular opponent or the player pool actually plays.

Entertainment: Mainly there to be watched rather than to teach, and not the host reviewing his own session. Typical formats: shows with other players such as pros against amateurs, reaction videos, travel and life stories, and big hands or famous moments told as a story.

Other: Not about playing poker better and not a poker show. Typical topics: channel news and announcements, coaching or course adverts, poker news, law and tax changes, interviews about life outside the game.

The "model", "state" and so on field names assume the camelCase JSON the repo uses; the C# records are JevRequest(Model, State, Questions) and JevChoiceQuestion(Type, Instructions, Criteria).

Editing the criteria doesn't change the content hash, so delete classified/ before you re-run classify. Otherwise Jev skips every video it has already sorted.




# 23 / 37

STEP 3 · CHUNK

## Cut on pauses, then add context

240 words · about 1:45 to read aloud

---

Step three: chunking. You cannot embed a forty-minute transcript as one vector; the meaning gets averaged into mush. So you cut it into chunks.

The naive way is every thousand characters, which cuts points in half. Captions give us better boundaries for free: a gap of four seconds or more starts a new paragraph. A pause is a decent proxy for a change of topic. Whole paragraphs are packed to about 400 tokens, overlapping by one paragraph, and each chunk keeps the start time of its first caption: that becomes the link to the second.

Then the glossary runs on every chunk, so "see bet" becomes "c-bet" before anything is embedded.

Then a line of context. Caption chunks do not say what they are about: "so he checks, and I'm thinking about it" could be anything. Claude Haiku reads each whole video, all its numbered chunks, and writes one or two sentences per chunk: hand review or concept, the spot, the opponent type, and the point.

This one is real. It is the video behind my corrections log, the rec who checks the river. Haiku's line names the spot and the point in plain poker terms, which the raw captions never do.

One call per video, not per chunk: about 380 calls instead of 14,000, no prompt cache needed. All of it cost $2.60, written once and kept. Where that line goes turned out to matter enormously. Hold that thought.






## One vector per chunk, in S3 Vectors

218 words · about 1:30 to read aloud

---

Step four: embed and store.

The embedding comes from IEmbeddingGenerator, the Microsoft.Extensions.AI interface. Bedrock appears in exactly one place, the DI registration, through AWS's AWSSDK.Extensions.Bedrock.MEAI package. The model is Amazon Titan Text Embeddings V2, 1,024 dimensions: cheap, the same call for questions and passages, and offered in London. Swapping it is configuration, measured with the eval you'll see in a few minutes.

The store is Amazon S3 Vectors: vector storage in S3, no cluster to run. PutVectors takes at most 500 vectors a call, so the writes are batched. Each vector's key is the video ID and chunk number. Bucket, channel, video and date are filterable; the text, title, start time and context line come back with each hit.

The rule that makes the weekly run boring: never write to the live index. Every build gets a new index, named to the minute it started. An SSM parameter points at the live one, with the model and dimensions it was built with. Embed moves that pointer only when no video failed, and keeps the newest three indexes, so a bad week rolls back by moving the pointer.

A full re-embed: 14,254 vectors, about 20 minutes, about 10 cents. Vector buckets made after 30 September 2026 are in ENHANCED mode by default, so filters run before the vector search.



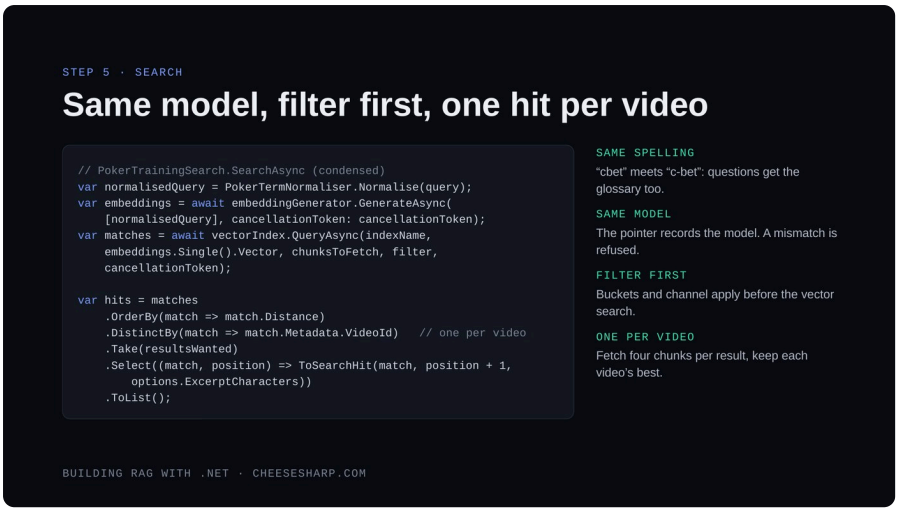

25 / 37

WHAT THE VECTOR DB RECEIVES

## One chunk, as S3 Vectors stores it

411 words · about 3:00 to read aloud


```
{
  "textToEmbed": "versus when a pro does it. So, let's jump in. Let me explain this first hand here. We're playing 25. We're playing against recreational Richard. He opens. We're going to defend on the button with king nine suited and we flop a flush draw. He c-bets. We are going to call. Turn is a blank. He's going to double barrel for a big size. We are going to call. Now the river is a deuce and recreational Richard checks to us on this river. One of the big mistakes I see even a lot of solid players make here when a recreational player checks. Just going for a medium or maybe even like a pot-size bluff here on this river. This is a huge mistake I think versus the player pool. When a recreational\n\nplayer bets the flop and bets the turn and then checks the river, it means something completely different than when a pro does it. In this case, most of the time when a recreational player checks the river, they are checking with the plan to check call. If you're going to bluff them on a river like this, you're going to have to choose an egregious size that they didn't expect. And if you've got this much money behind and they're not that sticky, maybe you can get them off of a hand like queen X or pocket kings or something like that. But just know when a recreational player checks the river here in general, they're not planning on folding. Betting 250 or betting 300 on this river is not\n\nFirst mistake, hand review at 2/5: a king-nine suited button defend vs a recreational player who c-bets, double barrels, then checks the river. The point is that a rec's river check is a check-call plan, so a medium or pot bluff is a mistake.",
  "putInputVector": {
    "key": "bx5YQWVtLSs#002",
    "data": { "float32": "[1,024 floats from amazon.titan-embed-text-v2:0, computed from textToEmbed]" },
    "metadata": {
      "bucket": "ExploitativeDirectTraining",
      "channel": "hungry-horse-poker",
      "videoId": "bx5YQWVtLSs",
      "text": "versus when a pro does it. … Betting 250 or betting 300 on this river is not",
      "title": "10 Mistakes Keeping Players Stuck at Low Stakes",
      "startMs": 72560,
      "published": "2025-08-19",
      "context": "First mistake, hand review at 2/5: a king-nine suited button defend vs a recreational player who c-bets, double barrels, then checks the river. The point is that a rec's river check is a check-call plan, so a medium or pot bluff is a mistake."
    }
  }
}
```






STEP 5 · SEARCH

## Same model, filter first, one hit per video

207 words · about 1:30 to read aloud

---

Step five: search. This is the whole of it.

First the question goes through the same glossary as the chunks, so "cbet" and "c-bet" meet in the middle.

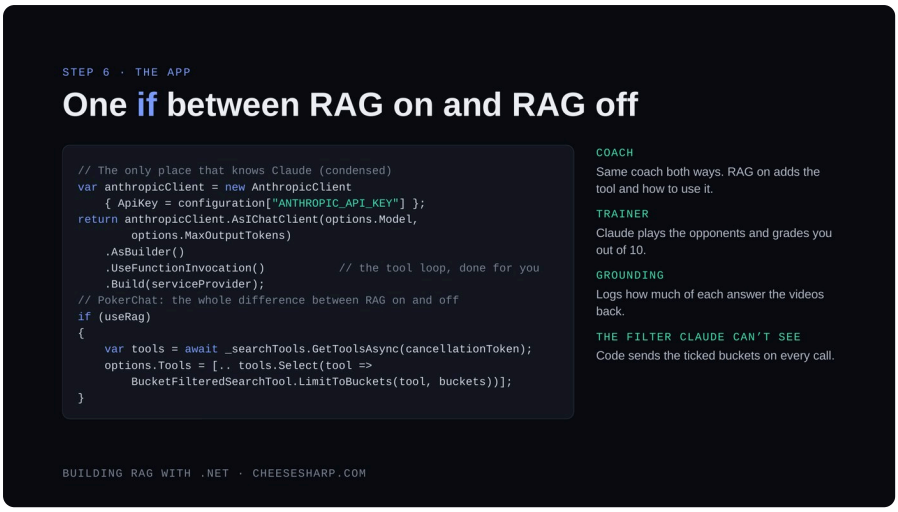
Then embed it with the same model as the index. Mixing models is the classic silent bug: everything runs, nothing matches, and you get confident nonsense. The live pointer records which model built the index, and search refuses a mismatch, per search, as an error Claude can see.

Then one query: the vector, how many chunks, and the filter. Buckets and channel apply before the vector search, so an unticked bucket cannot leak in.

One hit per video: without it, one long video filled the top five on its own. So search fetches four chunks per result wanted and keeps each video's closest. Five hits are five videos.

The live pointer is read on every search, so next week's index is picked up without a restart.

On top sits the MCP tool: one attribute, [McpServerTool], read-only, with structured output. Its description is written as a brief to a new starter, because that is how Claude decides when to search and which filters to use. It returns each excerpt with its title, channel, bucket and a link to the second.




# 27 / 37

STEP 6 · THE APP

## One if between RAG on and RAG off

287 words · about 2:00 to read aloud

---

Step six: the app. A Blazor Server app with three pages.

Coach: a chat with a RAG switch. Ask the same question with RAG off, and Claude answers from what it knows; with RAG on, it searches the two channels and cites them. Same expert coach both ways; RAG on adds the search tool and the instructions for using it.

Trainer: "in this spot, what would you do?" A live table, six to nine handed. The code deals and runs the preflop action; after the flop Claude Sonnet plays the opponents; Claude Opus grades every decision out of ten against what the coaches teach, with links to the moment.

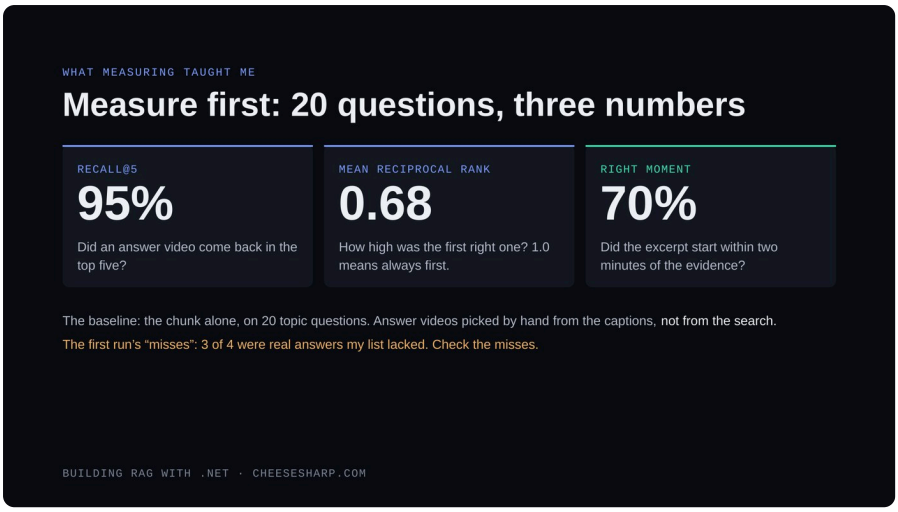
Grounding: every answer is logged, and this page shows how much of each answer the videos actually back. You will see why that page exists in a few minutes.

The code worth a slide. The official Anthropic C# SDK gives an IChatClient, the same Microsoft.Extensions.AI abstraction as the embeddings. UseFunctionInvocation is the whole tool loop: Claude asks for a search, it calls the MCP tool, sends the result back, and Claude carries on. No hand-written loop.

The MCP client's tools are already AIFunctions, so they plug straight into ChatOptions.Tools. And the whole difference between RAG on and off is one if.

One safety detail: Claude never sees the buckets parameter. A wrapper removes it from the tool's schema and sends the ticked buckets on every call, so evidence from an unticked bucket cannot reach an answer.

The chat runs Claude Opus 5.5 at high effort (the effort setting is the line I left out of the slide), about 8 to 12 cents an answer. Switching to Sonnet is one line of config: a decision to measure, not guess.






218 words · about 1:30 to read aloud

Here's the turn in the story. A RAG doesn't work the day it's assembled. It works after it's been measured and tuned, one stage at a time. So before the retrieval experiments, I built the ruler.

Twenty topic questions, each with the videos that answer it and a timestamped link to the evidence. Three numbers:

- Recall at five: did an answer video come back in the top five?
- Mean reciprocal rank: how high was the first right one? One over its rank, averaged. 1.0 means always first.
- Right moment: did the excerpt start within two minutes of the evidence? The right part of the right video, not just the video.

The answer videos came from searching the caption text, picked by me, not from the vector search. Grading the search with its own answers would prove nothing.

And the first run taught me to check the misses: three of the first four "misses" were real answers my own list didn't have. An eval built one way misses what another finds.

The recall step is one command and takes about six seconds. It writes into the run report next to everything else.

Video-level recall sat near 95% the whole time. The number that moved, and the one that matters to a student, was the right moment.




**29** / 37

WHAT MEASURING TAUGHT ME

## A famous recipe made it worse

273 words · about 2:00 to read aloud

---

Contextual retrieval: Anthropic's published recipe. Put a short model-written line in front of each chunk before you embed it. Their write-up shows big gains, and the chunks clearly needed it. My first build did that and added the video's title and channel too.

It made search worse. Recall at five dropped from 95% to 75%.

So I tested each part on its own. The recipe as published, the line in front of the chunk: 80%, still worse than the chunk alone. Titles alone hurt too: half are clickbait or "Ep 20", and stamping the same title on every chunk of a video makes its chunks look alike. Then the same line, moved after the chunk: back to 95%, the same reciprocal rank, and the right moment up from 70% to 75%.

Why? Put first, the line is a clean, dense summary in exactly the poker language the questions use, so it sets the direction of the vector. And the long "everything" videos, like "10 Years of Live Poker Advice in 3 hours", have dozens of chunks each summarised with every concept, so one of them lands near almost any question. After the chunk, the chunk's own words set the direction, and the line adds detail.

It won exactly where I needed it: specific moments buried in vlogs. "How wide do live pros 3-bet early" went from fifth to first.

Be honest about size: one question in twenty is within noise. This is "no worse, and better at the right moment", not a breakthrough.

The lesson: order mattered more than content. And a famous recipe can make your corpus worse. Measure it on yours.




30 / 37

WHAT MEASURING TAUGHT ME

## The “biggest single win” didn’t win

890 words · about 6:15 to read aloud

---

First, what reranking is. Vector search compares two vectors made separately: one for your question, one for each chunk, computed weeks apart. That's what makes it fast enough to search fourteen thousand chunks in a blink, and it's also why it's rough: neither vector ever saw the other text.

A reranker is a second, slower pass. Take the top 40 from vector search, then have a model read the question and each passage together and judge how well that passage answers that question. Think of a librarian who pulls forty books by the catalogue, then actually opens each one before handing you five. Cast a wide net cheaply, then judge carefully.

It's usually called the biggest single retrieval win, so I built one. Search fetches 40 candidate chunks, Claude Haiku 5.5 on Bedrock scores each from 0 to 10 against the question, and then search keeps the best chunk per video.

I also wrote a harder eval: twelve spot questions, concrete hands, the way the trainer asks. Plain vector search scored only 67% on those, against 90% on the topic questions, on the same index. An eval of easy questions hides the real weakness.

The reranker didn't help anywhere. Spot reciprocal rank went from 0.50 down to between 0.38 and 0.45.

Know your noise: the same reranker run twice moved the reciprocal rank by about 0.05. And look at what came back. For "c-bet out of position against a reg", Haiku put "Is It Worth Playing Online as a Live Poker Player? (Q&A)" and "10 Years of Live Poker Advice in 3 hours" first. The same trap as the context line in front of the chunk: broad content sounds related to everything.

Three seconds and a fifth of a cent per search, for nothing. It stays in the code, switched off by one setting. Next, if ever: a purpose-built reranker such as Cohere Rerank 3.5.

Two standard upgrades, two misses. That's what measuring is for.

### The columns

Spot and Topic are two different test sets.

Topic is 20 questions phrased like a video subject, such as "how to play against limpers".

Spot is 12 concrete hand situations, the way people actually ask in the trainer, such as "rec bets flop and turn, checks the river, I've got K9s…".

Both sets have answer videos you picked by hand.

R@5 (recall at 5) is the share of questions where at least one right video made the top five. On 12 spot questions, 67% is 8 of 12, 58% is 7 and 50% is 6. On 20 topic questions, 90% is 18 of 20 and 95% is 19.

MRR (mean reciprocal rank) is how high the first right video came. Each question scores 1 divided by that video's rank: 1st = 1.0, 2nd = 0.5, 3rd = 0.33, 5th = 0.2, and 0 if it's missing from the top 10. The scores are then averaged. Recall only asks whether the right video got into the top five; MRR also rewards putting it first. Roughly, 0.68 means the right video is usually first or second, and 0.50 means it's often second or lower.

### The rows

No reranker is plain vector search, the baseline.

Reranker, 1st run and 2nd run are the same reranker run twice on the same questions. Haiku scores 40 candidates 0 to 10 using the excerpt and its context line, then the top five are kept.

Reranker, no context lines is the same again, but Haiku doesn't see the context lines.

### How to read it

On spot questions the reranker was worse every time. Recall fell from 8 of 12 to 6 or 7, and MRR fell from 0.50 to between 0.38 and 0.45. The right moment was found less often and ranked lower.

On topic questions it made no real difference. Recall stayed at 18 of 20, apart from one run at 19 of 20, and MRR moved slightly either way. One question is within noise.

The two identical runs disagree with each other. Spot MRR was 0.38 then 0.42, and topic MRR was 0.61 then 0.66. That roughly 0.05 gap is pure noise from the model, so any difference smaller than that means nothing.

Spot 67% vs topic 90% in the baseline row is the other finding. Questions phrased like video titles flatter the search. The concrete questions people really ask expose its weakness.

Why it failed: Haiku favoured broad videos, such as "10 Years of Live Poker Advice in 3 hours", that sound relevant to almost anything, over the specific moment in a hand review. It's the same trap as putting the context line in front of the chunk.

Small samples: with 12 questions, one question moves recall by about 8 points, so be careful how you say it. The fair claim is: "no sign it helps, consistently a little worse on concrete hands, for 3 seconds and a fraction of a cent per search", not "it's broken".

On stage, about 20 seconds: "Two test sets: topic questions, and concrete hands the way people actually ask. Recall is 'did a right video make the top five'; MRR is 'how high was it'. The reranker never beat plain search, and it was worse on the concrete hands. And look at the two identical runs: they differ by 0.05 on their own. That's my noise floor. Anything smaller is nothing."






WHAT MEASURING TAUGHT ME

## Spot recall is the gap: 67% against a good 85%

365 words · about 2:30 to read aloud

---

Recall matters more than MRR for RAG. Claude reads all five excerpts, so it hardly matters whether the right one is first or third. What hurts is when the right video isn't in the five at all, because Claude can't cite what it never saw. Push recall up first and treat MRR as a tie-breaker.

"Too good to trust" means the test is too easy, not that the search is great. Your topic questions sound like video titles, which is why they score 90–95%. Your spot questions are the real test, and 67% is your gap. For this project I'd aim for:

Spot recall@5 of 80% or more (it's 67% now)

**Topic recall@5 staying at 90% or above**

**MRR of 0.6 or more on both sets**

MRR can't be much higher than recall. A question whose right video isn't in the top five scores 0.2 at most. A quick check is to divide MRR by recall@5:

**Topic: 0.68 / 0.90 ≈ 0.75**

**Spot: 0.50 / 0.67 ≈ 0.75**

The two are the same, so once the right video is found, it sits near the top either way. Spot questions fail because the right video isn't found, not because it's ranked badly. That points the fix at the question and chunk wording (glossary, rewriting the question before the search) rather than at reordering.

A change only counts if it's bigger than the noise:

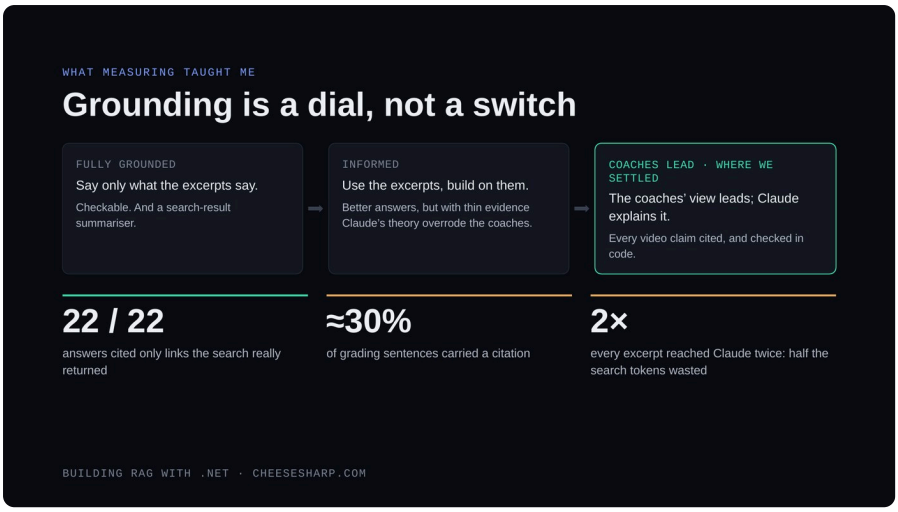
About ±0.05 MRR between identical runs.

One question moves spot recall by 8 points (1 in 12) and topic recall by 5 points (1 in 20).

To trust a 5-point change you'd need about 50 spot questions. That's the cheapest improvement available to you.

Cited sentences at about 30% has no standard either. For a coach, most advice sentences should cite a source, but the framing and summing up don't need to. So 50–70% would be a reasonable aim, and 30% is low.

For Q&A, if someone asks "is 0.68 good?": "Don't compare it to the leaderboards. Public tests like MS MARCO search millions of passages, and good systems score around 0.4 MRR there. Mine searches 385 videos with questions I wrote, so my numbers only mean something against my own earlier runs."




**32** / 37

WHAT MEASURING TAUGHT ME

## Grounding is a dial, not a switch

276 words · about 2:00 to read aloud

---

Earlier I said grounding means answering only from the evidence. Building it, I found grounding is a dial, not a switch, and I went too far before coming back.

Fully grounded: "say only what the excerpts say, cite every line." Checkable, and it turned Claude into a search-result summariser. It threw away the reasoning, the maths, adapting to your exact spot.

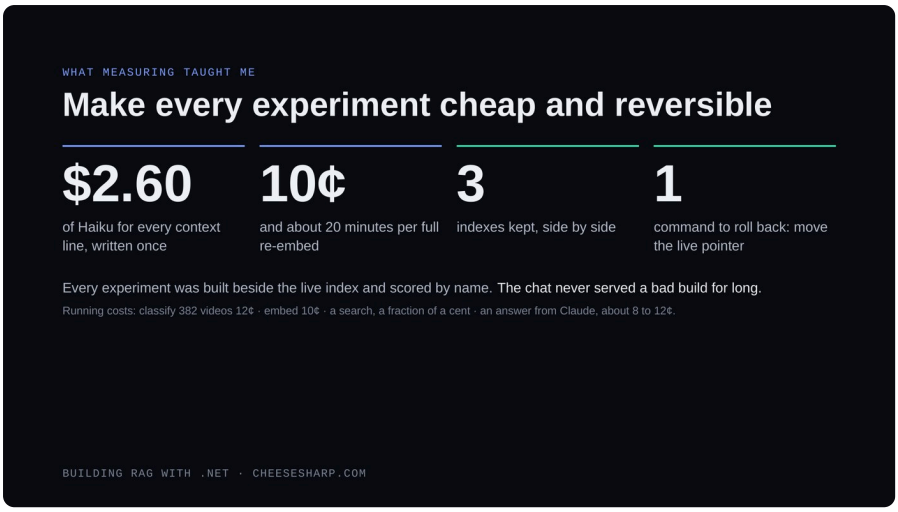
Informed: "use the excerpts as evidence and build on them." Better answers, but when the evidence was thin, Claude's general theory overrode the coaches. On a nut-changing turn it gave "my own opinion, check", against Hungry Horse, who teaches in about twenty videos to bet small there.

Where we settled: the coaches' view leads and is Claude's view, unless it's clear-cut wrong; Claude's analysis explains and extends it. Anything attributed to a video carries the excerpt's exact link to the second, and a GroundingCheck in code proves every cited link is one the search really returned. It can't prove the excerpt says what it's cited for; that's the honest limit.

The "what Claude saw" view splits every bad answer into one of two causes: retrieval missed, the clip isn't in the list; or generation ignored it, the clip is there, uncited. That split is the core of RAG debugging.

Then the first turn logs. All 22 answers cited only real links. Honest. But only about 30% of grading sentences carried a citation: Claude filled the rest itself. Grounding checks catch lies, not thinness. Measure both.

And reading the payload: every search sent each excerpt to Claude twice, as text and as JSON. About half the search tokens were waste. Read what the model actually receives.




**33** / 37

WHAT MEASURING TAUGHT ME

## Make every experiment cheap and reversible

162 words · about 1:15 to read aloud

---

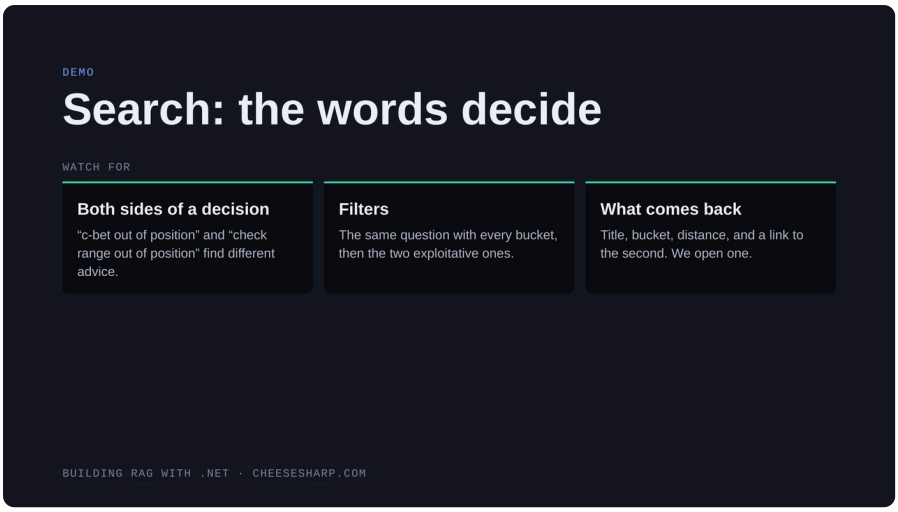
Last lesson, and the one that made the others possible: build so experiments are cheap and reversible.

Every experiment ran beside the live index. Embed builds a new index without switching the pointer; the recall step scores that index by name; only then do I decide. The chat never served a bad build for long, and rolling back is one command: move the pointer.

The context lines cost $2.60 of Haiku, once. Each re-embed costs about ten cents and twenty minutes. Classifying all 382 videos with Jev cost twelve cents. At this size, the cost of a RAG is engineering time and quality, not compute.

So the whole story: the first version found the right clips only when the question happened to use the right words. Every fix after that was found by looking at what the model saw and measuring the next change. Two famous upgrades didn't help. A thirty-line glossary fixed thirteen thousand phrases for free.

Now let's see it.




# 34 / 37

DEMO

## Search: the words decide

212 words · about 1:30 to read aloud

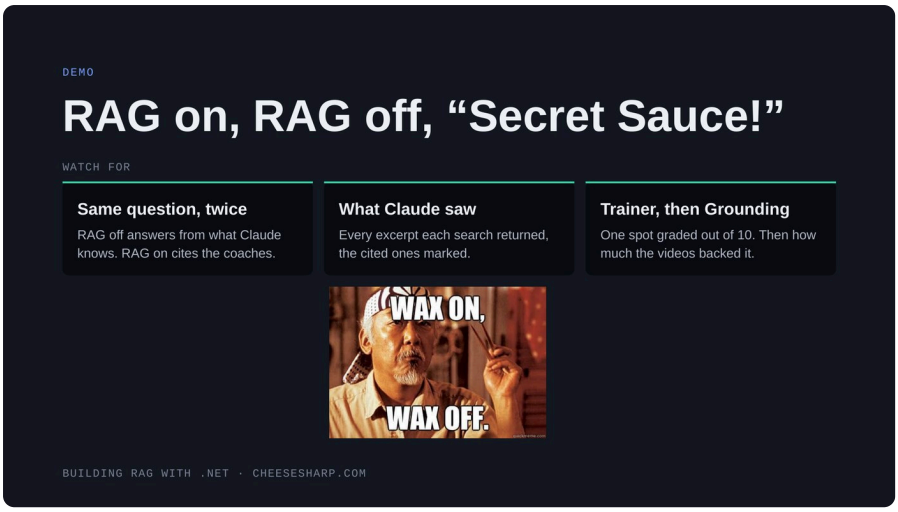
---

Goal: search on its own, no Claude yet, so the room sees the raw hits and how much the wording matters.

Terminal 1 is already running the MCP server: cd /mnt/d/Personal.git/poker-rag/rag-backend; dotnet run --project src/PokerRag.Mcp.Host (http://localhost:5080/mcp, as poker-rag-search). Open MCP Inspector: npx @modelcontextprotocol/inspector → Streamable HTTP, http://localhost:5080/mcp → Connect → Tools → search_poker_training.

1. (0:30) Show the tool and its description: that's what Claude reads.
2. (2:00) The wording decides. query "c-bet out of position against a reg", buckets ["ExploitativeDirectTraining","ExploitativeLiveHands"]. Then "check range out of position against recs". Hungry Horse's actual advice, "I'm a big fan of checking my range out of position because in general live players over-stab", only comes back for the second. Vector search matches meaning, so a question framed around one option finds evidence for that option.
3. (1:30) Filters: "how do I adjust against a maniac who raises every hand" with no buckets, then the two exploitative buckets. Point at anything from Entertainment or Other.
4. (1:30) Open one hit's link: the video starts at the right second. Point at the excerpt: normalised jargon, "c-bet" not "see bet".
5. (0:30) The structured content next to the text: the same hits, as JSON, for the app.

Backup: Claude Code connected with claude mcp add --transport http poker-search http://localhost:5080/mcp.




# 35 / 37

DEMO

## RAG on, RAG off, “Secret Sauce!”

199 words · about 1:30 to read aloud

---

Goal: why you'd put Claude on top, and how you check what it did.

Terminal 2: cd /mnt/d/Personal.git/poker-rag/rag-frontend; export DOTNET_ROOT=$HOME/.dotnet PATH=$HOME/.dotnet:$PATH; load ANTHROPIC_API_KEY and Rag__ObjectStore__LocalFolder from ~/.bashrc; dotnet run --project src/PokerRag.Web. Open http://localhost:5090.

1. (1:30) Coach, RAG off: "How should I adjust against a maniac who 3-bets every hand?" Claude's own knowledge, no citations. Switch RAG on: the question goes back in the box. Ask again. The amber search lines appear above the reply: Claude's own queries and the buckets.
2. (1:00) Click a search line: "what Claude saw". Every excerpt, distance, the cited ones marked in mint. "If the right clip isn't here, it's the search. If it's here and uncited, it's the prompt." Show the grounding badge: N citations from M excerpts, every one real.
3. (1:30) Trainer: deal one spot, make the decision, add a reason. Opponents play; the grade arrives: score, best action, a link to the moment. Ask "why not check?" on the card.
4. (1:00) Grounding page: the cited share per kind of answer, the verdict mix, and the least-grounded answer at the top. That's how I found "honest but thin".

If the Anthropic API is slow: Chat:Effort Medium, or play the recording.

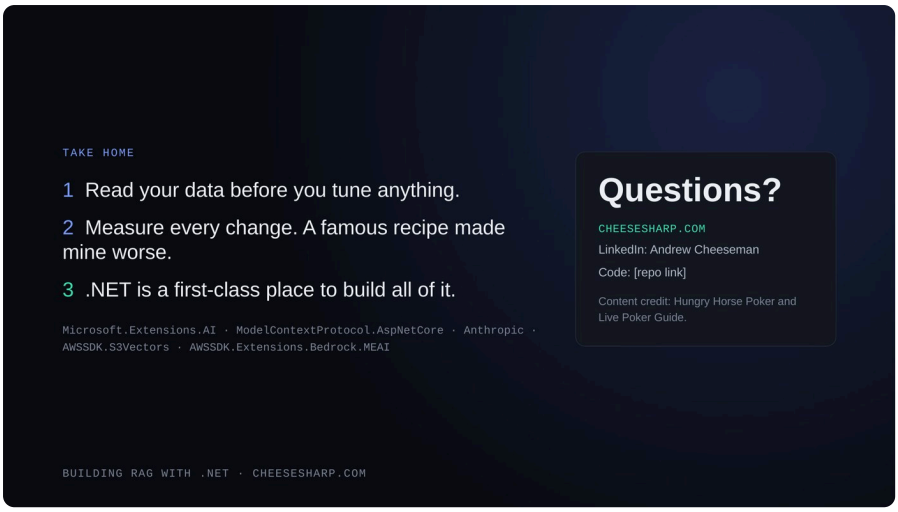



# 36 / 37

## Picture slide

0 words

*No speaker notes for this slide.*




# 37 / 37

## Questions?

346 words · about 2:30 to read aloud

---

Leave this slide up for the whole of questions.

Three things to take home: read your data before you tune anything; measure every change, because a famous recipe made mine worse; and .NET is a first-class place to build all of it. The packages are on the slide.

Thank Gavin and the venue. Credit both channels again.

Likely questions, with short answers:

- "Why not paste everything into a long-context model?" 14,254 chunks of about 400 tokens is millions of tokens: bigger than a context window, and retrieval wins on cost per question, freshness and citations.
- "Which embedding model?" Titan Text Embeddings V2, 1,024 dimensions, in London. Cheap and symmetric. Cohere might do better on short questions; it's a one-line change, measured with the same eval.
- "What does Jev cost, and why not just prompt an LLM?" 12 cents for 382 videos, input only. And a typed choice with a confidence and probabilities, instead of text to parse.
- "Why S3 Vectors?" No cluster, pay per use, filters before the search, pennies a month at this size. For hybrid keyword search or millisecond latency at scale, I'd look again.
- "Did contextual retrieval really not work?" As published, the line in front of the chunk, it hurt: 95% to 80%, and 75% with titles added too. The same line after the chunk matched the baseline and found the right moment more often. One question in twenty is noise; measure on yours.
- "Why not a reranker?" Built and measured: no gain, three seconds a search. Off by one setting. A purpose-built reranker is next, if retrieval needs it.
- "How do you know the answers are right?" The grounding check proves every cited link was really retrieved. It can't prove the excerpt says what it's cited for. Next: the API's citations, which quote the exact span, and a judge-model eval.
- "Is scraping YouTube allowed?" Personal study copy of public captions, not redistributed; excerpts only in answers. Check the terms for anything commercial.

*[Add the repo link if you publish the code. LinkedIn: linkedin.com/in/andrew-cheeseman-07a7527.]*