

BUILDING PRODUCTION MULTIMODAL AI AGENTS

Vision, Video, Screens, Documents,
Retrieval, Memory, and Evaluation



EVIDENCE EVENT		
SOURCE	camera-02	●
VALID TIME	00:14.220	●
REGION	x .42 y .18	●
PROVENANCE	sha256 + trace	●
SUPPORTED OBSERVATION		

VOICE IS ONE MODALITY. EVIDENCE
IS THE SYSTEM.

IA IAGENTIC LLC

ADRIAN MORGAN SEBULIBA

Founder & Lead Engineer, IAgentic LLC
iagentic.dev

Contents

Preface	5
1. The Frame Behind “This”	7
2. An Evidence Event, Not a Model Answer	11
3. Keep Each Modality Honest	15
4. Build the First World State	19
5. Pixels Are Not Objects	23
6. Ground the Claim	27
7. Read the Scene	31
8. Test What the Agent Sees	37
9. An Image Has No Before	41
10. Sample What Matters	45
11. Find the Moment	49
12. Resolve “This” While the World Moves	53
13. See the Screen, Read the Interface	57
14. Prefer Structure When It Exists	61
15. Turn Observations into Actions	65
16. Build a Safe Screen Agent	69
17. A Page Is More Than Its Text	73
18. Search Beyond Text	77
19. Cross-Modal Search	83
20. Hybrid Retrieval and Reranking	87
21. Multimodal RAG with Citations	91
22. Memory Is an Event Store	95
23. Ask the Past	99
24. Audio Beyond Speech	103
25. Sensors and World State	109
26. When Modalities Disagree	115
27. Evidence Before Action	121
28. Build the Multimodal Sandbox	127
29. Evaluate Every Boundary	133
30. Break It Deliberately	139
31. Design the Runtime	145
32. Build IAgentic Multimodal Studio	151
33. Deploy Without Losing the Evidence	157
34. Release with Tenant Boundaries	165
Appendix A: Services and Local Infrastructure	171
Appendix B: Reproducing the Book	173
Appendix C: References and Standards	175
Appendix D: Build-Along and Video Series Guide	181
Glossary	187
Support and Resources	189
About the Author	191

Preface

A voice agent hears a person say, “Remember where I put this.” The transcript contains no object and no location. The missing evidence is outside the words: a camera frame, a region in that frame, and the moment when the words were spoken.

That small problem is the foundation of this book. A production multimodal agent does not merely accept several file types. It constructs an evidence-backed world state from synchronized, sometimes contradictory observations, then acts without claiming more than those observations support.

The code and experiments proceed vertically. Every new modality must produce inspectable evidence, survive evaluation at its own boundary, and preserve its provenance when it joins the rest of the system. Failures remain in the story because they are part of the system being built.

The reader should be comfortable with Python, TypeScript, React, terminals, HTTP APIs and ordinary automated tests. The chapters introduce the additional geometry, media, retrieval and temporal reasoning as they become necessary.

Running the companion code

The code for the whole book lives in one companion repository:

<https://github.com/IAgentic-LLC/multimodal-agents>

```
git clone https://github.com/IAgentic-LLC/multimodal-agents
cd multimodal-agents
uv sync
uv run pytest -q
```

The Python package contains the evidence, perception, retrieval, memory, evaluation, policy, observability, and production API paths. The React and TypeScript Studio lives in `frontend/`:

```
cd frontend
npm ci
npm run build
npx playwright test
```

Docker Compose starts the production-shaped PostgreSQL, API, and Studio stack. Qdrant and Apache Tika have separate local service definitions for the search and document chapters. Exact commands and required environment variables are kept in the repository README and deployment files rather than copied into several chapters where they could drift apart.

The `runs/` directory contains the retained manifests, measurements, provider traces, and release-gate results cited by the book. Public fixtures and deterministic tests work without the author’s private credentials. Live provider, Auth0, and

cloud-deployment runs require the reader's own accounts and must never use credentials committed to the repository.

To verify the chapter currently being read, run:

```
uv run python scripts/verify_chapter.py CHAPTER_NUMBER
```

The repository and e-book both include [Reproducing the book](#), which maps all 34 chapters to their evidence gates and identifies every additional live service.

▫ Code availability

The e-book and repository are versioned together. If the link is temporarily unavailable during prerelease, use the edition's download page or contact the publisher rather than relying on an unofficial mirror.

1. The Frame Behind “This”

1.1 The problem

I want to ask a camera-equipped agent a simple question: “What am I looking at?” The sentence is incomplete on its own. “Looking at” points outside the transcript. The answer depends on a frame captured at the right moment.

A model can name the right object from the wrong frame. The sentence may sound correct while its provenance is false. Before choosing a vision model, I need a capture path that can prove which pixels and which spoken interval belonged to the same reference.

1.2 In plain words

The word “this” is a join across modalities. Speech supplies a referring expression. The camera supplies possible referents. Time narrows the join. Ground truth tells the evaluator which region I intended.

I therefore treat capture as an experiment, not as a preview window. A valid run retains the frame, bounded audio, one clock, device metadata, and an independently declared expected region. A later model answer cannot rewrite those inputs.

1.3 The first failure

The first browser run saved a valid JPEG and an empty audio artifact. Camera preview made the interface look healthy, yet the evidence bundle could not support a speech claim. The bundle exposed that failure before a model call could hide it behind a plausible answer.

I changed the recorder to construct a new `MediaStream` from the microphone track alone and asked `MediaRecorder` for data every 250 milliseconds. The W3C recording algorithm permits a user agent to impose a larger minimum slice, so the timeslice is a collection request, not a real-time guarantee. The code still waits for the recorder’s `stop` event before assembling the final blob.

1.4 Build the capture boundary

Camera and microphone access begins only after an explicit user action. The browser’s `getUserMedia` contract prompts for permission and either fulfills a promise with suitable tracks or rejects it. The UI represents both outcomes. It also exposes the selected physical devices instead of silently accepting a virtual camera left over from an earlier test.

The session uses `performance.now()` for offsets. That clock is monotonic and relative to `performance.timeOrigin`, so a wall-clock adjustment cannot make a later event appear earlier within the same browser session. I retain wall-clock

capture time too, but use it for audit and cross-system correlation rather than duration arithmetic.

The capture sequence is deliberately small:

1. obtain consent and start one camera-plus-microphone session;
2. snapshot the video frame when the reference begins;
3. record only the microphone track while the control is held;
4. stop and await the final audio data;
5. draw the expected object region on the retained frame;
6. save media and metadata under one capture identifier.

The operator declares the expected phrase and object before recording. That label is kept away from the perception model. Otherwise the evaluation answer would leak into the system being evaluated.

1.5 The retained run

The corrected live run retained a 1280 by 720 JPEG at a session offset of 717 milliseconds. The spoken interval began at the same offset and ended at 2,027 milliseconds. The JPEG contained 16,535 bytes, the Opus audio contained 20,511 bytes, and the JSONL metadata record contained 582 bytes.

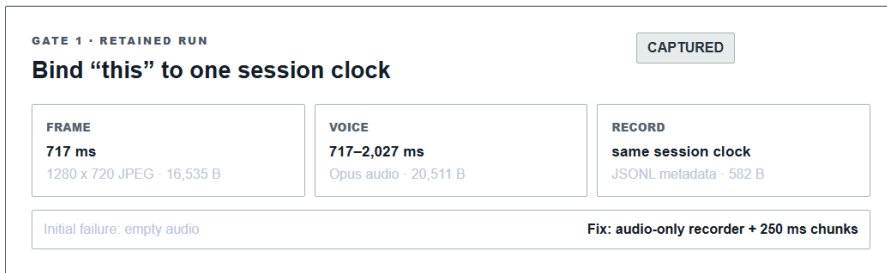


Figure 1.1: A retained capture panel showing a frame at 717 milliseconds, voice from 717 to 2,027 milliseconds, and one session-clock record. It also records the initial empty-audio failure and the audio-only recorder fix.

Figure 1.1 is an evidence display rather than an idealized architecture diagram. Each number comes from the retained Gate 1 artifact. The shared start offset does not prove perfect hardware-level synchronization. It proves the application associated one snapshot and one bounded utterance on its own session clock.

The capture does not yet prove what the speaker said, what object the frame contained, or whether a model can resolve the reference. Those are different measurements. Keeping that boundary explicit prevents a working recorder from being misreported as a working multimodal agent.

1.6 Test it live

The React and TypeScript interface has browser tests for the conditions that matter before intelligence enters the path. Playwright verifies that capture cannot begin without declared ground truth, a denied media permission returns the UI to a usable

state, and physical inputs are preferred over known virtual devices. The production frontend build is exercised too.

The retained run adds the check that unit tests cannot provide: real media was produced by the browser and inspected with media tooling. A non-empty file is still not proof of intelligible speech, but an empty file is decisive evidence that the pipeline failed.

1.7 Follow along

Episode 01 outcome. Capture synchronized camera, microphone, and monotonic timing evidence.

Work from the companion repository root. Open **frontend/src/App.tsx**, **frontend/src/capture.ts**, **src/multimodal_agents/capture.py**, **src/multimodal_agents/server.py**. Type the chapter’s focused listing yourself, then compare it with the complete implementation in those files; do not paste provider output or secrets into source control.

Run the smallest demonstrator:

```
uv run multimodal-capture
```

Then run the chapter contract:

```
uv run python scripts/verify_chapter.py 1
```

Expected evidence. Point at an object, speak, and inspect the paired frame, audio interval, and JSONL event. The verifier must finish with “chapter 1: tests passed” and validate **runs/gate-1**. **Environment:** Local browser; camera and microphone permission.

Episode checkpoint. Do not continue until the command, observable result, and retained artifact agree. Commit code only after removing local captures and credentials. The complete start-to-finish recording plan is in Appendix D, *Build-Along and Video Series Guide*.

1.8 What breaks

A single browser clock does not remove latency inside a camera, microphone, driver, or operating system. A frame may describe the scene a few milliseconds before or after its application timestamp. Long sessions can accumulate queue pressure. A virtual camera can be the intended source, and a physical camera can still show the wrong scene.

Privacy is part of the boundary too. Camera and microphone indicators must match actual track use. Ending the session stops every track and clears the preview. Raw media is retained only because this is an explicit evaluation run. Production retention needs purpose, access control, deletion, encryption, and tenant isolation rather than an indefinite “save everything” default.

1.9 Production pattern

Bind related observations with a monotonic session clock, but preserve their individual capture times and source identities. Retain exact inputs before provider

transformation. Require explicit consent, make device choice visible, and make stopping capture observable.

```
session
├─ frame + frame offset
├─ voice interval + audio asset
├─ declared expected phrase and object
├─ expected image region
└─ device and consent metadata
```

Chapter 2 turns this bundle into a durable application contract. Chapter 3 then demonstrates why sending both modalities to one model is not sufficient proof that either modality was interpreted honestly.

1.10 What this chapter did not measure

This run did not measure transcription quality, object recognition, grounding, model latency, model cost, long-session drift, hardware timestamp accuracy, or durable storage throughput. It proved that a live browser can produce a non-empty, inspectable, synchronized application-level bundle and that its first failure remains visible in the record.

Remember this

- “This” is a temporal join, not a complete text query.
- A correct object attached to the wrong frame has false provenance.
- Use a monotonic clock for within-session ordering and duration.
- Retain media before a provider transforms it.
- A healthy preview does not prove that audio was recorded.

2. An Evidence Event, Not a Model Answer

2.1 The problem

A model says, “The red mug is on the shelf.” That sentence may be correct, but the sentence alone is a poor unit of production state. It does not tell me which image supported the claim, when that image was captured, where the mug appeared, or whether a later observation superseded it.

The first architectural decision in this book is therefore not which model to call. It is what the system must retain after a model call ends.

2.2 In plain words

An **observation** is what a component claims. **Evidence** is the exact input that lets another person inspect the claim. An **evidence event** binds those two things to time, provenance, confidence, and relationships with other events.

I do not store this:

```
{"answer": "The red mug is on the shelf."}
```

I store enough to challenge it:

```
{
  "event_id": "evt-002",
  "session_id": "slice-001",
  "modality": "image",
  "event_type": "object_observed",
  "observation": {
    "object": "red mug",
    "location": "on the shelf"
  },
  "evidence": {
    "asset": "frames/frame-0087.jpg",
    "captured_at": "fixture-session-time",
    "region": {
      "left": 0.31,
      "top": 0.12,
      "right": 0.74,
      "bottom": 0.91
    }
  },
  "confidence": 0.94,
```

```
"session_offset_ms": 500
}
```

2.3 The picture

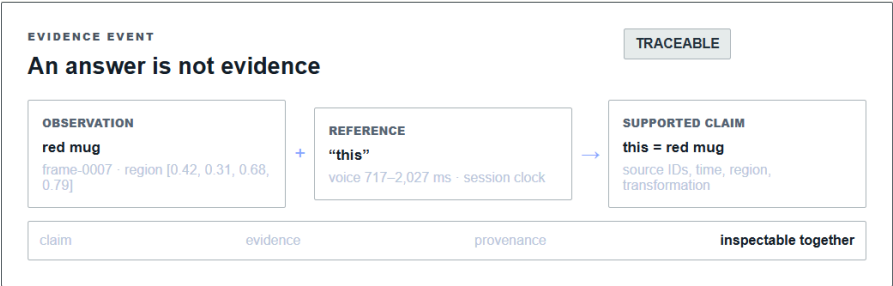


Figure 2.1: Three linked cards turn an observation and spoken reference into a supported claim while retaining source identifiers, time, region, and transformation provenance.

Figure 2.1 shows the minimum useful join. The claim remains inspectable because neither the observation nor the reference disappears inside the final sentence. The event has four layers:

claim	object = red mug, location = on the shelf
evidence	frame-0087.jpg, region (0.31, 0.12, 0.74, 0.91)
time	session offset 500 ms
provenance	event, session, modality and related event identifiers

The normalized region is not decoration. It is the difference between citing an entire image and citing the pixels that support the claim. Normalized coordinates also survive a change in display size, provided that later crops, rotations and padding preserve their transforms. Chapter 5 makes that qualification concrete.

2.4 Build it

The companion code makes invalid evidence difficult to construct. A `Region` must stay within zero and one and must have positive area. A `TimeRange` must start at or after zero and end after it starts. Confidence must stay between zero and one. Those checks do not prove that a region is correct. They prove only that the record is structurally capable of being evaluated. Correctness comes later, when a model region is compared with independently declared ground truth. The browser capture path creates another useful boundary. It retains the JPEG and bounded audio before either reaches a model. A later provider adapter may fail, change, or disappear without changing the input that was evaluated. That is why the media assets belong to the application, not to a transient model session.

2.5 The result

The current deterministic suite verifies that:

- invalid coordinates and intervals are rejected;
- a frame and audio interval are written under one capture identifier;
- an evidence event retains the cited asset and region;
- a wrong frame counts as a protected violation;
- an event can be serialized to JSONL and recovered later.

The distinction between ordinary errors and protected violations matters. An object label can be wrong and still be inspectable. A correct object label attached to the wrong frame creates false provenance. This book treats that as a more serious class of failure.

2.6 Follow along

Episode 02 outcome. Define and validate the evidence-event contract.

Work from the companion repository root. Open `src/multimodal_agents/evidence.py`, `src/multimodal_agents/capture.py`. Type the chapter’s focused listing yourself, then compare it with the complete implementation in those files; do not paste provider output or secrets into source control.

Run the smallest demonstrator:

```
uv run pytest -q tests/test_vertical_slice.py
```

Then run the chapter contract:

```
uv run python scripts/verify_chapter.py 2
```

Expected evidence. Reject malformed regions and time ranges before they enter world state. The verifier must finish with “chapter 2: tests passed” and validate `runs/gate-0` and `runs/gate-1`. **Environment:** Deterministic local replay.

Episode checkpoint. Do not continue until the command, observable result, and retained artifact agree. Commit code only after removing local captures and credentials. The complete start-to-finish recording plan is in Appendix D, *Build-Along and Video Series Guide*.

2.7 What breaks

The schema does not make a model truthful. A model can assign 0.99 confidence to the wrong region. A caller can label the wrong ground-truth box. A file can be replaced after its event was written. Two processes can use clocks that cannot be compared.

Those are separate controls:

- hashes and immutable storage protect asset identity;
- independent labeling protects evaluation truth;
- one monotonic session clock protects duration and ordering;
- wall-clock time supports audit and correlation across systems;
- provider and model provenance make a result reproducible.

The first implementation does not claim all five controls yet. It establishes the record that later chapters will strengthen.

2.8 Production pattern

Treat model output as a proposed observation. Promote it into world state only with inspectable evidence and explicit provenance. Keep the evidence even when the proposed observation is rejected, because the rejection is itself useful evaluation data.

2.9 What this chapter did not measure

This chapter did not measure model accuracy, region quality, transcription, latency, cost, retention durability, or storage throughput. It proved the shape and deterministic validation of the evidence boundary.

Remember this

- A sentence is not evidence.
- A confidence number is not provenance.
- A correct label on the wrong frame is a protected violation.
- Retain the exact input before a provider transforms it.

3. Keep Each Modality Honest

3.1 The problem

The first live capture produced an unexpected result. The retained audio was noisy, while the camera feed visibly contained mirrored DroidCam text. A joint image-and-audio request returned the transcript “inverted text,” localized that same visible text, and assigned confidence 0.95.

That looked coherent. It was also not reliable evidence of what the speaker said.

3.2 In plain words

When one request contains several modalities, agreement inside the answer can be circular. The image can influence the supposed audio transcript, then the same image can make that transcript appear visually grounded.

The system needs to distinguish useful cross-modal reasoning from one modality silently rewriting another.

3.3 The picture

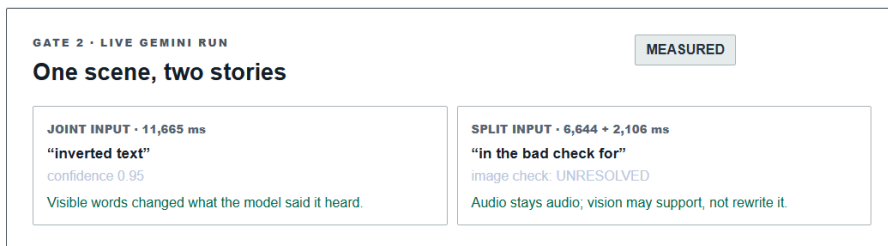


Figure 3.1: Two measured paths through the same retained media. The joint Gemini request reports the visible words “inverted text” as speech with confidence 0.95, while the split path preserves an independent audio transcript and returns unresolved when vision cannot support it.

Figure 3.1 uses the actual Gate 2 outputs and timings. It does not suggest that the split transcript is correct. It shows that the split path preserves disagreement rather than letting visible text rewrite audio.

The unsafe path asks one component to establish every fact:

```
audio + image
      |
      v
```

```

joint model
|
+--> transcript
+--> object
+--> region
+--> confidence

```

The protected path creates separate evidence-producing stages:

```

audio --> transcription ----+
                             |
image + transcript --> grounding --> supported or unresolved

```

The second stage may use the transcript as a query. It may not edit the transcript to make the image easier to explain.

3.4 Build it

I retained one JPEG and one Opus/WebM clip, then ran three checks against the same media:

1. a joint request received both audio and image;
2. an audio-only request received only the clip;
3. the split path passed the isolated transcript to an image-only grounding request.

The joint adapter remains in the repository because a bad baseline is useful when it is reproducible. It is not the default production path.

The split adapter returns an unresolved result when the spoken query is not supported by the image. That result is not patched into a guess. Abstention is an output the evaluator can reward when the reference is absent.

3.5 The result

The joint request returned “inverted text.” Two audio-only passes returned “and Vivace four” and “in the bad check for.” None of the three transcripts agreed.

FFmpeg found a valid mono 48 kHz Opus stream with non-zero signal. That proves the clip was not empty. It does not prove that the signal contained intelligible speech.

The split grounding stage received the second isolated transcript and the same JPEG. It returned `resolved: false`. The joint path took 11,665 ms. The two split stages took 6,644 ms and 2,106 ms.

My first cost calculation was wrong. I counted visible candidate tokens but not the model’s thinking tokens, even though the pricing contract bills both. I fixed the adapter and reran the same retained media. The joint rerun used 1,205 input tokens, 114 candidate tokens and 1,095 thinking tokens. At the retained standard-price snapshot, that is \$0.0054375. The split rerun used 1,204 input tokens and 648 billed output tokens across both stages, or \$0.003333. The corrected comparison reverses the original cost ordering.

The rerun reproduced the behavioral distinction. The joint path again turned visible mirrored text into its transcript and grounded it with confidence 0.95.

The split transcription returned “il vivace qua,” and the grounding stage returned `resolved: false`.

This is one observed case. It is not a model-wide error rate. Its value is that the retained inputs and incompatible outputs expose a failure that a single coherent-looking response would have hidden.

3.6 Follow along

Episode 03 outcome. Separate provider output from normalized observations.

Work from the companion repository root. Open `src/multimodal_agents/gemini.py`, `src/multimodal_agents/run_grounding.py`. Type the chapter’s focused listing yourself, then compare it with the complete implementation in those files; do not paste provider output or secrets into source control.

Run the smallest demonstrator:

```
uv run python -m multimodal_agents.run_grounding
```

Then run the chapter contract:

```
uv run python scripts/verify_chapter.py 3
```

Expected evidence. Compare raw provider evidence with the normalized event. The verifier must finish with “chapter 3: tests passed” and validate **runs/gate-2**. **Environment:** Live Gemini run requires `GOOGLE_API_KEY`; retained replay does not.

Episode checkpoint. Do not continue until the command, observable result, and retained artifact agree. Commit code only after removing local captures and credentials. The complete start-to-finish recording plan is in Appendix D, *Build-Along and Video Series Guide*.

3.7 What breaks

Separating stages costs another request and can discard useful information. Audio and vision really do inform each other in ordinary human perception. A strict split is therefore not a claim that modalities should never interact. It is a control for facts whose origin matters.

The split can also fail independently. A poor transcription gives grounding the wrong query. An image model can still localize the wrong object. A stale frame can be perfectly grounded and temporally false.

The evaluator must score transcript, object, frame, region and abstention separately. One aggregate “correct” flag would hide which boundary failed.

3.8 Production pattern

Use joint multimodal models to propose relationships. Use isolated or independently checkable stages for claims that authorize actions, enter long term memory, or appear in an audit trail. Preserve disagreement instead of forcing the modalities to tell one story.

3.9 What this chapter did not measure

The run used one noisy clip and one placeholder camera frame. It did not measure performance on deliberate speech, a real object, a population of speakers, multiple models, the Live API, or repeated trials. Those limits are why the result is a reproduced failure case rather than an accuracy estimate.

Remember this

- A coherent multimodal answer can still have circular support.
- Self-reported confidence did not expose the disagreement.
- Non-zero audio is not proof of intelligible speech.
- `unresolved` is safer than a visually convenient transcript.
- A cost trace that drops thinking tokens can reverse a comparison.

4. Build the First World State

4.1 The problem

An observation becomes useful after the moment that produced it. If I ask, “Where is the red mug?” the agent must find the latest supported observation, not the first matching sentence and not whichever event was appended last.

If I ask, “What changed?” one observation is not enough. The honest answer is that the system lacks history.

4.2 In plain words

A world state is not one mutable dictionary that claims to contain reality. It is a view computed from evidence events. The events remain immutable. The view can change when later evidence arrives or when a contradiction is found.

4.3 The picture

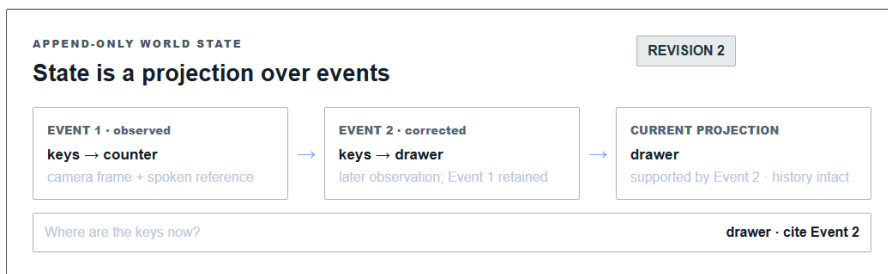


Figure 4.1: An append-only sequence retains an original keys-on-counter observation and a later keys-in-drawer correction, then derives a current drawer projection citing the later event without erasing history.

Figure 4.1 separates events from the current projection. The current answer can change without rewriting the evidence that supported a previous answer.

The deterministic scenario contains two events:

```
100 ms  red mug observed on the desk  cites evt-001 / frame-0042
500 ms  red mug observed on the shelf  cites evt-002 / frame-0087
```

The answer to “where?” needs the second event. The answer to “what changed?” needs both.

4.4 Build it

`EvidenceMemory` stores newline-delimited event records. `where_is` filters by object and orders matches by monotonic session offset. It returns the latest location plus an evidence citation.

`what_changed` requires two observations. It returns both citations and a description of the transition. If only one observation exists, it returns `insufficient_history`. If none exists, `where_is` returns `insufficient_evidence`.

The important code is not the sentence template. It is the refusal boundary:

```
if not matches:
    return TemporalAnswer(None, (), "insufficient_evidence")
```

The result separates the human-readable answer, machine-readable status and supporting citations. A UI can phrase the answer differently without changing what supports it.

4.5 The result

The clean deterministic run printed:

```
object correct: True
frame correct: True
region IoU:     1.00
memory found:  True
violations:     0
where:          red mug was last observed on the shelf.
where cites:    evt-002
changed:        red mug moved from on the desk to on the shelf.
change cites:   evt-001, evt-002
```

Ten Python tests pass. They include the supported answer, insufficient history, unknown object, event ordering, wrong-frame violation and region IoU. The frontend separately passes four Playwright tests and a production build.

4.6 Follow along

Episode 04 outcome. Reduce immutable evidence events into queryable world state.

Work from the companion repository root. Open `src/multimodal_agents/memory.py`, `src/multimodal_agents/world_state.py`. Type the chapter's focused listing yourself, then compare it with the complete implementation in those files; do not paste provider output or secrets into source control.

Run the smallest demonstrator:

```
uv run pytest -q tests/test_vertical_slice.py
```

Then run the chapter contract:

```
uv run python scripts/verify_chapter.py 4
```

Expected evidence. Ask for the latest supported object location without mutating source evidence. The verifier must finish with “chapter 4: tests passed” and validate runs/gate-0. **Environment:** Deterministic local replay.

Episode checkpoint. Do not continue until the command, observable result, and retained artifact agree. Commit code only after removing local captures and credentials. The complete start-to-finish recording plan is in Appendix D, *Build-Along and Video Series Guide*.

4.7 What breaks

The latest event is not automatically the truest event. A later camera may be occluded. A sensor may disagree. Two events from different session clocks may not be comparable. An object name may refer to two different mugs.

This first state view uses exact object equality and one session clock on purpose. Entity identity, cross-session time, contradiction and evidence quality appear later. Adding those concerns now would hide the small contract this run actually proves.

4.8 Production pattern

Keep observations append-only and compute current state as a cited view. Make absence of evidence and absence of history explicit statuses. Never turn a missing citation into a fluent guess.

4.9 What this chapter did not measure

The scenario is a deterministic fixture. It does not establish database throughput, semantic entity matching, long-term retention, cross-session ordering, model accuracy, or multi-user isolation. It proves temporal query semantics and evidence-preserving answers.

Remember this

- World state is a view over evidence, not an overwritten truth object.
- “Latest” must use a comparable session clock.
- A change claim needs at least two cited observations.
- Missing history should remain missing.

5. Pixels Are Not Objects

5.1 The problem

A model can return the right rectangle and an application can still draw it over the wrong object. The failure often happens after inference. The model saw source pixels, while the interface displays a resized, letterboxed, or cropped version of them.

Suppose the source is 1920 by 1080 and the viewport is 640 by 640. Copying a normalized source box directly into that square silently assumes that the source and viewport have the same aspect ratio. They do not.

5.2 In plain words

A region is not just four numbers. It belongs to an image, a coordinate convention, an orientation, and a chain of transforms. Lose any one of those facts and the box can remain numerically valid while becoming visually false.

5.3 The picture

Figure 5.1 shows the same source region rendered with two common fit modes. The dotted red box is the tempting shortcut: use the source coordinates without transforming them. The green box is mapped through the actual scale and offset.

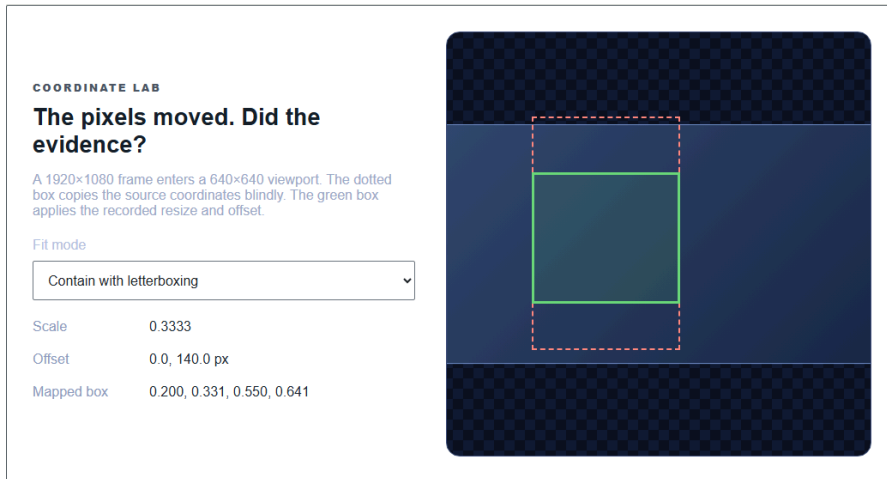


Figure 5.1: A live-tested coordinate lab compares contain and cover rendering. In both panels a red dotted box copies source coordinates directly, while a green box uses the scale and offset of the rendered image. The contain panel includes letterboxing and the cover panel includes cropping.

5.4 Build it

The geometry layer stores the source and target sizes plus the scale and offset selected by the fit operation. For `contain`, the complete source must fit inside the target:

```
scale = min(target.width / source.width,
            target.height / source.height)
```

For `cover`, the target must be filled, even if part of the source is clipped:

```
scale = max(target.width / source.width,
            target.height / source.height)
```

The rendered width and height determine the remaining offsets. A normalized source point (x, y) maps into target pixels as:

```
target_x = offset_x + x * source_width * scale
target_y = offset_y + y * source_height * scale
```

Only after that mapping should the result be normalized against the target. The inverse operation subtracts the offset before dividing by scale. It must also distinguish source content from padding. A click in a letterbox is not a point in the source image.

The implementation represents a rectangle as left, top, right, and bottom in normalized coordinates. That choice is not universal. COCO commonly stores left, top, width, and height in pixels. Other APIs use corner points or integer grids. Conversion happens at an explicit boundary rather than inside an unrelated drawing component.

Orientation is another boundary. Camera and phone images may carry EXIF orientation metadata. The pixels must be transposed before regions are interpreted, or the transform history must state that they were not.

5.5 The result

For a 1920 by 1080 source displayed in a 640 by 640 target, the live React and TypeScript lab mapped the same source box differently:

Fit mode	Mapped target region
contain	(0.200, 0.331, 0.550, 0.641)
cover	(0.000, 0.200, 0.589, 0.750)

The cover result begins at the left edge because part of the source lies outside the visible crop. Clipping it is correct. Pretending the hidden area is visible would produce a false citation.

Python tests cover both fit modes, round trips, clipping, and rejection of a region entirely inside padding. Playwright drives the TypeScript interface, changes the fit mode, and checks the displayed coordinates. This matters because a correct Python evaluator cannot protect a separate browser implementation from drifting.

5.6 Follow along

Episode 05 outcome. Convert safely among pixel, normalized, and display coordinates.

Work from the companion repository root. Open `src/multimodal_agents/geometry.py`, `src/multimodal_agents/run_geometry.py`, `frontend/src/geometry.ts`. Type the chapter’s focused listing yourself, then compare it with the complete implementation in those files; do not paste provider output or secrets into source control.

Run the smallest demonstrator:

```
uv run python -m multimodal_agents.run_geometry
```

Then run the chapter contract:

```
uv run python scripts/verify_chapter.py 5
```

Expected evidence. Map a model region onto a resized UI without shifting the overlay. The verifier must finish with “chapter 5: tests passed” and validate `runs/gate-3/coordinate-results.json`. **Environment:** Deterministic local run.

Episode checkpoint. Do not continue until the command, observable result, and retained artifact agree. Commit code only after removing local captures and credentials. The complete start-to-finish recording plan is in Appendix D, *Build-Along and Video Series Guide*.

5.7 What breaks

Coordinate bugs reappear when a pipeline:

- resizes an image but retains boxes from the original dimensions;

- applies `object-fit: cover` without retaining the crop offset;
- draws normalized values against the wrong viewport;
- rotates pixels using EXIF metadata after inference;
- confuses left-top-width-height with corner coordinates;
- rounds early and repeatedly; or
- clips a region without recording how much evidence became invisible.

A particularly dangerous response is to clamp every number into the interval from zero to one. Clamping can make malformed output look valid. Validate the source region first, transform it second, and clip only at a declared display boundary.

5.8 Production pattern

Treat every region as an evidence pointer with an image identifier and a transform record. Keep source coordinates canonical. Derive display coordinates for a specific viewport, and test the inverse mapping. If the visible crop does not contain the cited object, mark the citation unavailable instead of drawing a persuasive box over nearby pixels.

5.9 What this chapter did not measure

This gate proves coordinate behavior for deterministic fixtures and two CSS fit modes. It does not measure model localization accuracy. It also does not yet test real EXIF-rotated camera files, arbitrary perspective transforms, or long preprocessing chains.



Remember this

- Four numbers are not a region without their coordinate contract.
- `contain` introduces padding; `cover` introduces cropping.
- Keep canonical source coordinates and derive viewport coordinates.
- A box in invisible pixels cannot support a visible claim.

6. Ground the Claim

6.1 The problem

An agent that always returns a box will eventually box something that is not there. Positive-only demos reward confident localization and conceal the more important behavior: refusing an invalid visual reference.

Grounding therefore has two tasks. It must decide whether the requested object is present, and, only when present, identify the supporting region.

6.2 In plain words

The answer to “Where is the green triangle?” is not the nearest colorful shape. If the image contains only a red rectangle and a blue circle, the correct result has no region.

Absence is a first-class outcome, not a zero-sized box and not a low-confidence guess.

6.3 The picture

The retained fixture in Figure 6.1 contains one planted positive and one planted negative query. The model region for the red rectangle coincides with the known fixture region. For the absent green triangle, the adapter emits no box.

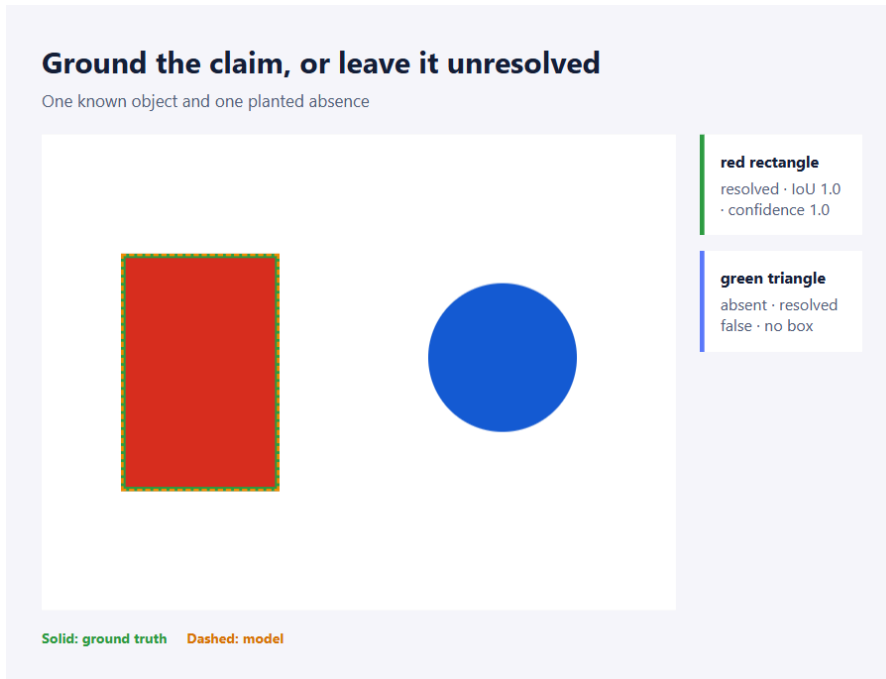


Figure 6.1: A retained white test image contains a red rectangle and blue circle. The ground-truth and model boxes coincide on the red rectangle. Result cards show IoU 1.00 for the present query and an absent result with no box for a green triangle.

6.4 Build it

The fixture is generated in a browser at exactly 640 by 480 pixels. Its red rectangle occupies pixels 80 through 240 horizontally and 120 through 360 vertically. The normalized region is therefore:

```
{
  "left": 0.125,
  "top": 0.250,
  "right": 0.375,
  "bottom": 0.750
}
```

Each case declares whether the reference is present. A present response must resolve the requested object and supply a valid region. An absent response must return `resolved: false` and no region. The adapter removes placeholder coordinates from an unresolved provider response before scoring it.

Intersection over union measures spatial agreement for positive cases:

```
IoU = area(prediction intersect truth)
      / area(prediction union truth)
```

The evaluator retains the continuous value and can also report whether it meets a threshold. It does not compute IoU for an absent case because there is no true rectangle. Instead, it counts a protected violation if the system forces a region onto an absent reference.

The provider trace also retains latency and usage. The accounting includes candidate and thinking tokens because both contribute to billed output for the model used in this run. That detail corrected an earlier experiment in which candidate tokens alone understated cost.

6.5 The result

The two retained calls produced these outcomes:

Query	Expected	Result	Region IoU	Latency
red rectangle	present	resolved	1.00	3,770 ms
green triangle	absent	unresolved	not applicable	1,427 ms

The positive call used 1,127 input tokens and 746 billed output tokens. At the captured standard price of \$0.75 per million input tokens and \$3.75 per million output tokens, its estimated cost was \$0.00364275. The absent call used the same input count and 145 billed output tokens, for an estimated cost of \$0.001389.

Neither case produced a protected violation. The positive region matched the fixture exactly. The negative response explicitly named the visible red rectangle and blue circle while declining to invent the triangle.

These are two cases, not an accuracy estimate. Their purpose is to validate the data contract, evaluator, accounting, overlay, and absent-reference path against a real provider response.

6.6 Follow along

Episode 06 outcome. Require claims to carry regions and source evidence.

Work from the companion repository root. Open `src/multimodal_agents/image_grounding.py`, `src/multimodal_agents/run_image_grounding.py`. Type the chapter’s focused listing yourself, then compare it with the complete implementation in those files; do not paste provider output or secrets into source control.

Run the smallest demonstrator:

```
uv run python -m multimodal_agents.run_image_grounding
```

Then run the chapter contract:

```
uv run python scripts/verify_chapter.py 6
```

Expected evidence. Draw the cited region and distinguish observation from inference. The verifier must finish with “chapter 6: tests passed” and validate `runs/gate-4`. **Environment:** Live Gemini run requires `GOOGLE_API_KEY`; retained replay does not.

Episode checkpoint. Do not continue until the command, observable result, and retained artifact agree. Commit code only after removing local captures and

credentials. The complete start-to-finish recording plan is in Appendix D, *Build-Along and Video Series Guide*.

6.7 What breaks

Grounding systems fail when they:

- require a region even when the reference is absent;
- treat $(0, 0, 0, 0)$ as a legitimate absent box;
- score negative cases with positive-only IoU;
- trust confidence without checking object identity;
- draw model coordinates without the transform from Chapter 5.;
- retain a prose description but discard the actual frame; or
- report a threshold pass while hiding poor continuous IoU.

Synthetic fixtures are useful because ground truth is exact, but they are easy. Natural scenes add occlusion, small objects, ambiguous language, multiple instances, text, reflections, and motion. A production benchmark needs those cases plus invalid references.

6.8 Production pattern

Separate reference resolution from localization. Make **resolved** explicit. Require a region only for resolved positive claims. Cite the exact retained image and preserve the coordinate contract. Evaluate present and absent cases with different metrics, then combine them in a report that cannot hide forced matches.

6.9 What this chapter did not measure

Two deterministic shapes cannot establish provider accuracy, calibration, or robustness. This run does not compare models, prompt variants, natural-image datasets, repeated-call variance, or adversarial references. It establishes a reproducible gate that later experiments can extend without changing what counts as an honest answer.

Remember this

- Grounding begins by deciding whether the reference exists.
- An absent object has no region.
- Preserve continuous IoU and count forced matches separately.
- Two successful cases validate a protocol, not a model.

7. Read the Scene

7.1 The problem

OCR can recover every word in a screenshot and still destroy the scene. A flattened transcript does not say which values belong to a table, which marks form a chart, where either element appeared, or whether two visual elements describe the same measurements.

For an agent, that missing structure changes what can be proved. The string `Checkout 410` is useful. A typed table cell at a cited region is stronger. A matching bar in a separate chart can corroborate it, but only if the relationship is retained rather than guessed later from nearby words.

7.2 In plain words

Reading a scene means preserving both content and layout. Text answers what was written. Regions answer where it was written. Element types answer what role it played. Relationships answer how one part supports another.

7.3 The picture

I built a deterministic incident dashboard containing a title, a two-row latency table, and a bar chart of the same values. Figure 7.1 shows the regions returned by the real model call. The types and relationship are correct, but the boxes are deliberately shown as returned. The title region absorbs the subtitle, and the other regions are looser than the planted truth.

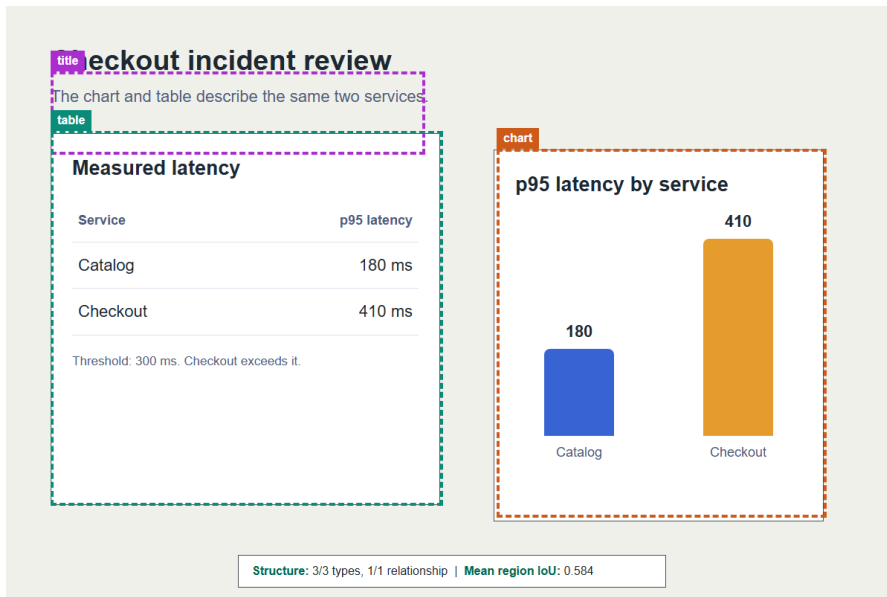


Figure 7.1: A dashboard screenshot is overlaid with dashed model regions for its title, table, and chart. The title box extends into the subtitle. A result strip says that all three types and the one relationship were recovered, while mean region IoU was 0.584.

7.4 Build it

The browser fixture has a fixed 1200 by 800 viewport. The table contains Catalog at 180 milliseconds and Checkout at 410 milliseconds, plus a 300-millisecond threshold. The chart repeats the two service values as bars.

The extraction contract keeps typed elements:

```
@dataclass(frozen=True)
class SceneElement:
    element_id: str
    kind: str
    text: str
    region: Region
```

It also keeps relationships separately:

```
SceneRelation(
    "latency-chart",
    "visualizes_same_measurements_as",
    "latency-table",
)
```

That shape follows a pattern shared by current document systems. Tesseract's TSV output can retain a box and confidence for each word instead of returning plain text alone. Google Document AI represents tables, figures, typed layout blocks,

hierarchy, and bounding polygons. Its layout documentation explicitly describes flattened OCR as destructive for retrieval. Gemini structured outputs then let this experiment require a predictable JSON schema.

The evaluator checks four things independently:

1. element-kind recall;
2. continuous region intersection over union;
3. relationship recall; and
4. unsupported relationships.

The flattening baseline is intentionally simple. It joins element text and demonstrates what disappears: types, coordinates, and the `visualizes_same_measurements_as` edge. This is a representational loss, not a claim that plain OCR never has a useful role.

7.5 The result

The real Gemini 3.8 Flash call returned all three requested IDs with the correct types and recovered the single supported chart-to-table relationship. It introduced no unsupported relationship.

Measure	Retained result
Element-kind recall	1.000
Relationship recall	1.000
Unsupported relationships	0
Mean region IoU	0.584
End-to-end latency	23,117 ms

The call used 1,155 input tokens, 368 visible candidate tokens, and 5,227 thinking tokens. Billed output was therefore 5,595 tokens. At the captured standard prices used in the preceding experiment, the estimated call cost was \$0.0218475.

The result is instructive. Content extraction and structural classification were correct on this fixture. Spatial precision was much weaker. A system that scored only OCR strings would declare success and conceal the loose regions.

7.6 Follow along

Episode 07 outcome. Extract a structured scene graph from visual evidence.

Work from the companion repository root. Open `src/multimodal_agents/scene_reading.py`, `src/multimodal_agents/run_scene_reading.py`, `frontend/src/OpeningEvidenceLabs.tsx`. Type the chapter's focused listing yourself, then compare it with the complete implementation in those files; do not paste provider output or secrets into source control.

Run the smallest demonstrator:

```
uv run python -m multimodal_agents.run_scene_reading
```

Then run the chapter contract:

```
uv run python scripts/verify_chapter.py 7
```

Expected evidence. Inspect objects, relations, confidence, and provenance together. The verifier must finish with “chapter 7: tests passed” and validate **runs/gate-5**. **Environment:** Live Gemini run requires `GOOGLE_API_KEY`; retained replay does not.

Episode checkpoint. Do not continue until the command, observable result, and retained artifact agree. Commit code only after removing local captures and credentials. The complete start-to-finish recording plan is in Appendix D, *Build-Along and Video Series Guide*.

7.7 What breaks

Scene readers fail when they:

- flatten table rows and lose cell membership;
- transcribe chart labels but ignore the visual encoding;
- detach a figure from its caption or heading;
- return a page-sized box for every element;
- infer a relationship merely because two blocks are adjacent;
- accept repeated values as proof that two visuals share a source; or
- discard the screenshot after producing JSON.

Tables and charts also disagree in less tidy material. Units may differ. An axis can be truncated. A chart may aggregate rows that remain separate in the table. A screenshot can be stale while an API is current. The world state must preserve those as separate observations before deciding whether they corroborate or contradict one another.

7.8 Production pattern

Keep the original asset. Extract text into typed blocks with regions. Preserve table cells, chart series, captions, headings, and explicit edges as separate records. Score content, structure, geometry, and relationships independently. Use plain text as one searchable projection of that richer record, not as the record itself.

7.9 What this chapter did not measure

This is one synthetic dashboard and one provider call. It does not establish OCR error rates, chart-reasoning accuracy, repeated-call variance, handwriting support, multilingual quality, or performance on dense documents. It proves that the structured contract is runnable and that a content-correct result can still have measurable spatial error.

 Remember this

- OCR text is a projection of a scene, not the scene itself.
- Preserve types, regions, hierarchy, and relationships.
- Score spatial precision even when every word looks correct.
- Keep the original screenshot as the final evidence boundary.