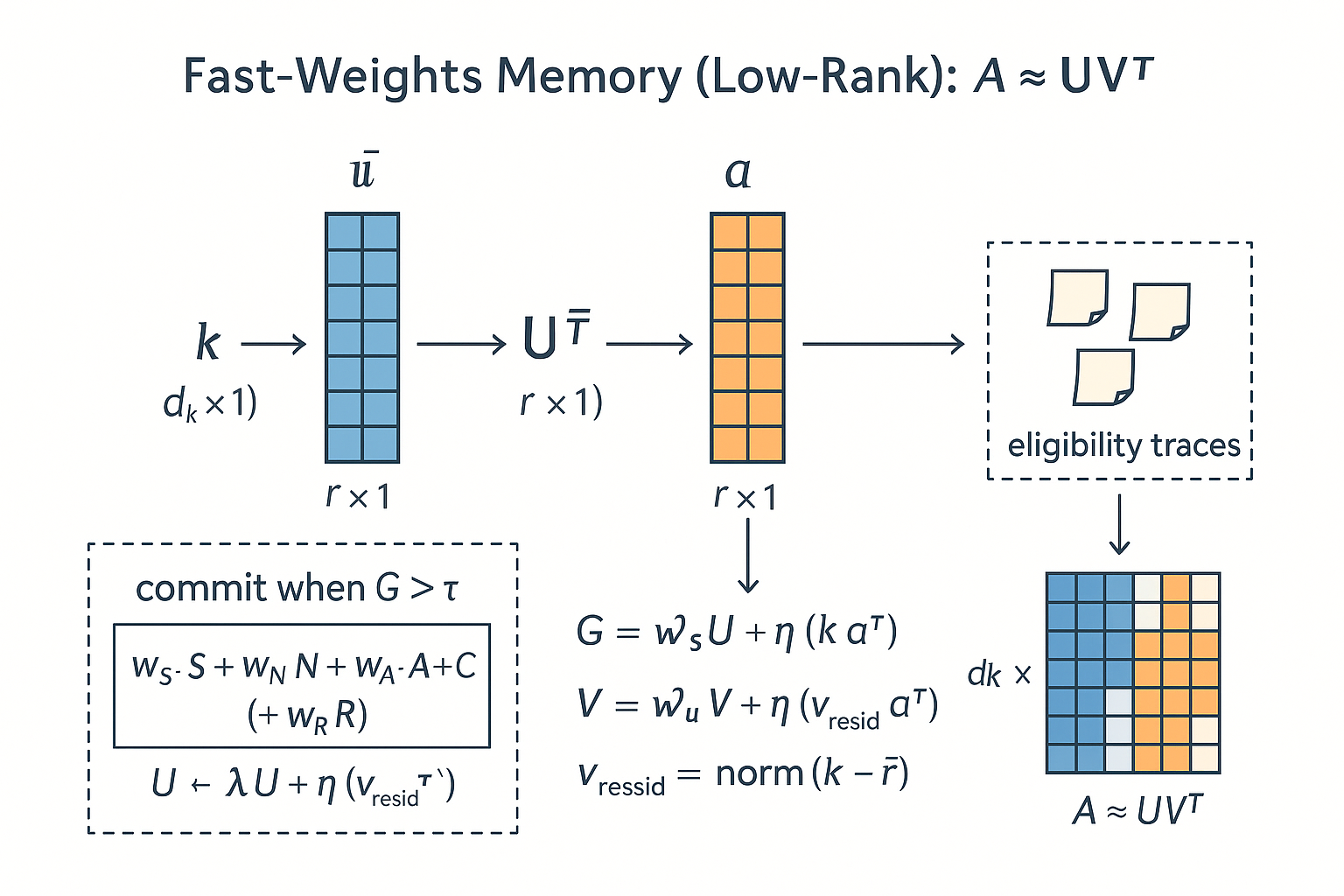
# **Sidecar Memory — Plain‑English Overview (v0.3.3)**



A small add‑on that helps AI systems **remember the right things** across conversations—without retraining the model or stuffing the prompt. It works with common Hugging Face models (like GPT‑2 and Llama‑style LMs) and keeps memory tiny, fast, and under your control.

## **What it is**

**Sidecar Memory** is a plug‑in memory module that sits next to your language model. Think of it like a smart notebook and a helpful librarian:

* The **notebook** stores compact notes about past moments.
* The **librarian** only writes things down when they seem important and pulls the right note at the right time.

## **What problem it solves**

* **Selective recall:** Remembers key facts and moments (preferences, changes, warnings) instead of every word of the chat.
* **No model changes:** Works without fine‑tuning or modifying the base model’s weights.
* **Compact footprint:** Memory size stays **constant**, even as chats get long.
* **Stable answers:** Reduces “drift” and improves consistency for facts you’ve told it before.

## **How it decides what to remember**

Humans don’t remember everything—we remember what stands out. Sidecar does the same, using a simple rule: **only write if it’s worth it.** It looks for:

* **Surprise:** Unusual phrases or sudden shifts.
* **Novelty:** New information compared to what it already knows.
* **Arousal (importance):** Signals like warnings (e.g., “thermometer flashed a warning”), extreme numbers (e.g., **117°F**), or named places (e.g., **Death Valley**).
* **Conflicts:** If you say something that **replaces** an earlier fact (e.g., “Actually, my favorite color is red”), it notes the change.

You can also explicitly tell it to remember something (e.g., “Remember that my deadline is Friday”).

## **What it actually stores**

Rather than copying your sentence, it stores a **compact fingerprint of what was new** (a tiny vector). That keeps memory small and helps it generalize across similar situations.

## **How it helps during a reply**

1. It quickly checks memory for relevant notes.
2. It uses them to **steer** the reply (e.g., include your preferences).
3. For short, exact answers (like a color or date), it can run a **deterministic recall** step to answer reliably.

## **Where it shines (examples)**

* **Personalization:** “My favorite color is blue.” → Later: “What color did I say I like?” → **blue**.
* **Safety/Operations:** “Thermometer flashed a warning at 117°F.” → Flag and remember high‑risk events.
* **Customer memory:** Preferences, past issues, recent changes.
* **Agent workflows:** Track ongoing tasks or decisions between sessions.

## **Privacy & control**

* **Opt‑in facts:** You can choose what gets stored.
* **Erase anytime:** Delete specific notes or wipe all memory.
* **Portable:** Save to disk and reload later; easy to audit.

## **What it *doesn’t* do (current limits)**

* It’s not a full database; it stores **high‑value highlights**, not transcripts.
* Arousal/importance uses simple cues (warning words, extreme numbers, notable places). It’s effective but intentionally lightweight.
* For complex structured data (e.g., calendars, multi‑field profiles), pair it with a proper store.

## **Status & quick results**

* In tests, routine text like “I ate a sandwich; it was fine” **isn’t stored**, but “Drove through **Death Valley** in **117°F** heat; the **thermometer flashed a warning**” **is**—just like a human would remember the striking moment.
* Simple fact teaching (e.g., “Remember my favorite color is blue”) is **recalled correctly** even after restarting the session.

## **Roadmap (what’s next)**

* **Reward hook:** Let users mark “this is important” for guaranteed storage.
* **Images:** Add a vision path so pictures can trigger the same kind of memory.
* **Consolidation:** Periodically compress older notes to keep memory fresh.

## **One‑sentence pitch**

**Sidecar Memory gives your AI a human‑like sense of what’s worth remembering—keeping the important stuff at its fingertips, without retraining or bloat.**

**Short version**

* A hard drive (or vector DB/RAG log) **stores everything and grows forever**.
* Our sidecar **stores only what matters** and stays **constant-size**—two tiny low-rank matrices that self-update when the surprise/affect gate fires.
* Reads/writes are **tiny math ops** (no prompt bloat, no re-embedding big chunks), so latency stays flat.

**Concrete numbers (apples to apples)**

* Typical vector DB item (1536-d, fp16) ≈ **3 KB** each.  
  + 100k items ≈ **~293 MB** (before index overhead).
* Sidecar fast-weights (dₖ=256, rank=64, fp32):  
  + U (256×64) + V (256×64) + self-state ≈ **~130 KB total**.
* That’s roughly **~2,300× smaller** than a 100k-item vector store (and still **~23,000× smaller** than 1M items).
* Compute: sidecar read/write ≈ **O(dₖ·r)** (here, 256×64), i.e., a few tens of thousands of multiplies—essentially constant.

**Why this beats “just bolt on a hard drive”**

* **Selectivity vs. hoarding:** We write only when the event is novel/surprising/important (and can be user-rewarded). A drive logs everything indiscriminately.
* **On-manifold memory:** We store a compact **residual in the model’s key space**, so recall blends in naturally. A drive stores raw text or embeddings you must re-rank, re-inject, and pay token costs for.
* **Flat latency:** Sidecar lookup is constant and returns a tiny vector; RAG needs ANN search + chunk fetch + prompt stuffing.
* **No drift from long prompts:** We don’t balloon the context window; answers stay stable.
* **User control:** Keep/forget policies, reward boosts, cold-start guard—without retraining or touching base weights.

**When a hard drive *is* better**

* Audit/compliance, full transcripts, analytics over raw conversation history. (We play nice with that: keep your archive, let the sidecar be the **working memory**.)

# 

# **Sidecar Memory — Technical Details (v0.3.3)**

Affect‑gated, low‑rank fast‑weights memory that bolts onto any Hugging Face–style causal LM (no base‑weight changes).

## **1) Design Goals & Constraints**

* Selective persistence across sessions **without** fine‑tuning the base LM.
* **Constant memory footprint** (low‑rank fast weights) regardless of dialogue length.
* **Minimal surgery**: prepend a tiny textual prefix; write/read via a sidecar; optional KV recall for deterministic answers.
* **Human‑like gating** using surprisal, novelty, and affect/arousal; optional conflict & reward boosts.
* **Device‑safe & portable** (CPU/GPU) with simple save/load.

## **2) Architectural Overview**

**Core components**

1. SelfStateManager — tiny GRU that tracks a persistent self‑state s used for steering and key binding.
2. FastWeightsMemory — low‑rank matrices U and V that implement a Hebbian fast‑weight store over a key space of size d\_k.
3. Eligibility Traces — short‑window queue that binds temporally adjacent events when a later surprise occurs.
4. AffectScorer — lightweight arousal estimator (lexicon + numeric outliers + NE priors + punctuation/intensifiers).
5. SurpriseGate — blends signals into a gate score G compared against an adaptive threshold tau (with cold‑start guard).
6. StatePrefixGenerator — maps s to a compact, steerable textual prefix.
7. KV Recall — constrained decoding that injects a small memory text into the KV cache and decodes from a restricted lexicon.

**Data flow (per request)**

* Encode prompt → key k. Read fast weights → reconstruction r. Update self‑state with r. Assemble prefix + user text. Compute surprisal/novelty/arousal/conflict → gate. If G > tau → commit residual; else push an eligibility trace. Generate response.

## **3) Mathematical Formulation (Plain‑Text)**

### **3.1 Keys, Reads, Residuals**

* Hidden feature (last token): h in R^(d\_m).
* Key projection: k = norm( tanh(W\_q \* h) ) in R^(d\_k).
* Read mixture over low‑rank factors:  
  + a = softmax( transpose(U) \* k / T ) # shape r
  + r = transpose(a) \* V # shape d\_k
* Residual to store: v\_resid = norm( k - r ).

### **3.2 Surprise, Novelty, Affect, Conflict**

* Novelty: N = (1 - cos(k, r)) / 2 (range 0..1).
* Surprisal S: token‑level negative log‑likelihood (NLL) on the assembled text; squash to 0..1 via S = tanh(NLL / 3).
* Arousal A (0..1): AffectScorer(text) using lexicon + numeric outliers + named‑entity priors + punctuation/caps.
* Conflict C: equals 1.0 when a new fact contradicts a known one (e.g., latest favorite color differs), else 0.0.
* Reward R: optional user‑supplied importance signal in 0..1 (hook available).

### **3.3 Gate & Threshold**

* Gate score: G = wS\*S + wN\*N + wA\*A + wC\*C + wR\*R.
* Threshold tau: either fixed (fixed\_tau) **or** adaptive using an EMA of G with variance; during cold start use warmup\_tau for the first min\_events updates.
* Commit rule: write **only** if G > tau (strict inequality).

### **3.4 Hebbian Update (Low‑Rank Fast Weights)**

* Decay: U = lambda\*U and V = lambda\*V (0 < lambda < 1).
* Mixture coefficients: a = softmax( transpose(U) \* k / T ).
* Write gain: eta = eta0 \* G.
* Updates:  
  + U = U + eta \* ( k \* transpose(a) )
  + V = V + eta \* ( v\_resid \* transpose(a) )
* Stability: normalize each column of U and V to unit L2 norm after the update.

### 

### **3.5 Eligibility Traces**

* If not committed, enqueue (k, v\_resid, S) with TTL (time window) w seconds. On the next commit within the window, also commit these traces with the current gain.

## **4) Algorithms (Pseudocode)**

**WRITE(text)**

k = encode\_key(text)

S, N, r = surprise\_novelty\_read(k, text) # NLL + novelty, returns reconstruction r

A = affect(text) # arousal in [0,1]

C = conflict(text, known\_facts)

G = wS\*S + wN\*N + wA\*A + wC\*C + wR\*R

tau = threshold() # EMA or fixed; cold‑start guarded

if G > tau:

commit(k, norm(k - r), gain=G)

for (k\_e, v\_e, \_) in elig.pop\_alive():

commit(k\_e, v\_e, gain=G)

else:

elig.add(k, norm(k - r), S)

update\_known\_facts(text)

**GENERATE(prompt)**

assembled = prefix(self\_state) + prompt

WRITE(assembled) # updates gate/eligibility, maybe commits

return LM.generate(assembled)

**KV RECALL (deterministic)**

# Build KV from a short memory text only (no user echo)

past = LM(mem\_text, use\_cache=True).past\_key\_values

logits = LM(prompt, past\_key\_values=past).logits[-1]

if allowed\_words: mask logits to that lexicon

argmax step for a few tokens → decode → post‑clean one word

## **5) Complexity & Footprint**

* Storage: U, V are d\_k x r each → O(d\_k\*r) parameters (constant w.r.t. session length).
* Read: matrix‑vector transpose(U) \* k → O(d\_k*r); reconstruction also O(d\_k*r).
* Commit: two rank‑1 outer products → O(d\_k\*r) + per‑column normalization.
* KV recall: constant‑time prefill + greedy decode for a few tokens.

Example small config: d\_k = 256, r = 64 → about 33k floats per matrix (≈256 KB total in fp32; less with fp16/bf16).

## **6) Public API Surface**

SidecarV3(model, tok, device="cpu", d\_key=256, d\_state=512,

rank=64, key\_topk=32, decay=0.995, eta0=0.1, temp=0.5,

affect\_weights=(0.3, 0.15, 0.5, 0.05, 0.0),

fixed\_tau=None, min\_events=2, warmup\_tau=0.90,

s\_mode='blend', s\_alpha=0.6)

.generate(prompt, max\_new\_tokens=64, temperature=0.7, top\_p=0.9)

.generate\_kv(mem\_text, prompt, max\_new\_tokens=16,

stop\_on\_newline=False, allowed\_words=None)

.save\_state(path) / .load\_state(path)

**Key parameters**

* affect\_weights=(wS, wN, wA, wC, wR) — blend of signals into G.
* min\_events, warmup\_tau — cold‑start guard for thresholding.
* rank, key\_topk — capacity & sparsity; separation in key space.
* eta0, decay, temp — write learning rate, forgetting, read temperature.
* s\_mode, s\_alpha — how S and N are combined.

## **7) Tuning Guide**

* Too many commits early? Increase warmup\_tau (e.g., 0.95–0.99), raise wA, or lower wS if NLL is noisy.
* Missing salient events? Raise wA and wN, or shorten the eligibility window.
* Drift / overwrite? Lower eta0 or increase decay; reduce rank to encourage reuse rather than overwrite.
* Fact recall ambiguity? Use generate\_kv with an allowed\_words lexicon.

## **8) Evaluation Protocols**

**Scenarios**

1. Ordinary vs extreme: “sandwich was fine” vs “Death Valley 117°F warning.” Expect commit only for the latter.
2. Fact learning/recall: “favorite color is blue” → one‑word recall via KV; persistence across reload.
3. Conflict handling: say blue, then red → commit with C boost; latest answer is red.

**Metrics**

* Commit precision/recall vs a human‑labeled salience set.
* Hit@1 for fact recall (KV and open‑gen).
* Memory stability under long dialogues (sweep rank, decay, eta0).
* Latency overhead vs base LM.

## **9) Failure Modes & Mitigations**

* Gibberish with high NLL → add fact‑pattern filters before committing; rely more on A/N via affect\_weights.
* Numeric outlier false positives → tighten unit heuristics; cap A contribution; maintain recent numerical baselines.
* Topic churn causing overwrite → increase rank or decay; lower eta0; enable eligibility traces.
* Sensitive data capture → add redaction rules or a permission prompt; support forget() and state wipes.

## **10) Compatibility & Ops**

* Works with HF causal LMs (e.g., GPT‑2/NeoX/Llama‑family) on CPU or CUDA; no model surgery.
* Safe with mixed precision; writes run under torch.no\_grad().
* save\_state(path) writes self\_state.pt, fastweights.pt, and gate.json.

## **11) Extensibility**

* Reward/RPE hook (R): user‑supplied importance boosts; log‑scaled or binary.
* Vision key‑path: CLIP/ViT features → keys; visual arousal from contrast/anomaly/scene hazard.
* Better affect: small classifier or emotion embeddings instead of heuristics.
* Consolidation: periodic replay to compress traces into the low‑rank store (or an episodic long‑term store).
* Structured memory: slots for preferences, tasks, identities with conflict‑aware merges.

## **12) Changelog (recent)**

* v0.3.3: Arousal channel (A), conflict boost (C), strict gate, improved KV lexicon mask.
* v0.3.2: Strict G > tau, novelty clamp to [0,1], early cold‑start guard; KV recall mask.
* v0.3.1: NLL/novelty blend; KV recall tweak; stats show tau.

License: MIT