BRAIN_SCAN Client User Manual

Version 2.0 — covers SDK v1.1.0 and the current service

1. What BRAIN_SCAN does, in plain words

BRAIN_SCAN watches your AI agent and learns what “normal” looks like for it. After a short teaching period, it stands guard: every time your agent acts, BRAIN_SCAN compares the new behaviour against everything it was taught. If the behaviour looks normal, nothing happens. If it looks unusual, BRAIN_SCAN raises a flag and puts the event in a review queue for you.

You then make the final call:

- **“This is fine”** — the new behaviour is added to what BRAIN_SCAN considers normal.
- **“This is bad”** — it stays on record as a threat, and BRAIN_SCAN keeps watching for it.

Nothing is ever learned automatically while guarding. Only you can teach BRAIN_SCAN new behaviour after the guard is on. This is deliberate: it means no attacker can slowly train the system to accept bad behaviour.

The three words you will see everywhere

Word	Plain meaning
Learn	Teaching mode. You send examples of normal behaviour.
Arm	“Lock the doors.” Teaching ends, thresholds are set, guarding can begin.
Guard	Watching mode. Every window of behaviour is checked: MATCH (normal) or OUT_OF_DISTRIBUTION (unusual).

Two more words, explained once

- **Engram** — a “behaviour fingerprint.” When you teach, BRAIN_SCAN does not store your raw data. It compresses every 24 events into one compact fingerprint. A typical teaching run produces dozens of these fingerprints, and together they form your agent’s library of normal.
- **OUT_OF_DISTRIBUTION (OOD)** — simply means “unusual for THIS agent.” It does not automatically mean an attack. New kinds of normal work also trigger it, which is why the review queue exists.

2. Before you start

You need four things:

1. **An API key** — we give you this. It looks like a random string of letters and numbers. Keep it secret; it is the password to your agents’ libraries.
2. **A name for your agent** — any short label, e.g. `"support-bot"`. One library is built per name.
3. **Python 3.8 or newer** on the machine where your agent runs.
4. **Your agent’s activity as events** — one event is one thing your agent did, written as a small JSON dictionary. Example:

```
{"timestamp": "2026-08-01T09:15:00Z",
  "request_id": "req-1042",
  "tool_name": "search",
  "arguments": {"query": "shipping rates", "limit": 5}}
```

One important preparation step: events usually contain fields that change every single time, like `timestamp` and `request_id`. These are noise to BRAIN_SCAN — they make two identical actions look different. Make a list of those field names now. You will pass them once as `ignore_keys`, and BRAIN_SCAN will drop them before learning.

3. Installing the SDK

Run this once:

`pip install` https://api.azoulaye.com/dist/brainscan-1.1.0-py3-none-any.whl

The command must end in the wheel's filename — pip decides what it is downloading from the address exactly as typed. (The shorter address https://api.azoulaye.com/dist/brainscan redirects here for humans, but is not itself a valid install target.)

Check it worked:

```
import brainscan
 print(brainscan.__version__)   # should print 1.1.0
```

4. Connecting

```
from brainscan import BrainScan

scan = BrainScan(
     api_key="paste-your-key-here",
     agent_id="support-bot",
     ignore_keys=["timestamp", "request_id"],   # your noisy fields
 )
```

If you prefer not to put the key in code, set these environment variables and write `scan = BrainScan()` instead:

```
BRAIN_SCAN_API_KEY=your-key
 BRAIN_SCAN_AGENT_ID=support-bot
```

Good habits from the start:

- Never commit the key to a shared repository.
- Set `ignore_keys` on day one. It is saved permanently with your first teaching call and cannot be changed afterwards without starting over.

5. The golden rule of teaching

Read this section even if you read nothing else.

Your first teaching call sets the measuring stick for everything that comes after.

When BRAIN_SCAN receives your agent's very first learn call, it builds a fixed "measuring stick" (the reference frame) from everything inside that one call. Every future measurement — every teach, every guard scan, for the whole life of that agent — uses that same stick.

So the first call must contain a **representative bundle** of normal behaviour: several days or weeks of typical operation, not one small slice. If the first call is too thin, the stick is crooked, and healthy behaviour will look unusual forever. The only repair then is to delete the agent and teach it again correctly.

The easy way to do it right — `learn_curriculum()`:

```
scan.learn_curriculum([week1_events, week2_events, week3_events])
```

Pass it several windows of normal behaviour (each window is a list of event dictionaries). It pools them into ONE first call.

Two things people worry about, answered:

- *"Will pooling squash my library into one fingerprint?"* No. The engine still creates one fingerprint per 24 events. A pooled seed of ~1,600 events produces about 70 fingerprints with the full detection cascade intact.
- *"Can I keep teaching after the first call?"* Yes. Streaming with `send()` after the seed is exactly right. The golden rule applies only to the very first call.

The service watches your back. If your first call looks too thin, the response includes a warning list with code **BS-W2001**. If your events arrive on the wrong detection channel, you get **BS-W2002**. These are warnings, not errors — the teaching still completes — but treat them as a stop sign: delete the agent (Section 10) and re-teach correctly before you arm. See Section 12 for details.

6. The full journey, step by step

Step 1 — Teach

Collect several windows of normal behaviour (for example, one window per day for a week or two; a few hundred events each), then:

```
result = scan.learn_curriculum([day1, day2, day3, day4, day5])
 print(result)
```

A healthy response looks like this (your numbers will differ):

```
{"status": "LEARN_COMPLETE",
  "engrams_added": 69,        # fingerprints created
  "raw_engrams_total": 69}
```

Check for `protocol_warnings` in the response. If you see any, stop and read Section 12 before continuing.

Adding more normal behaviour later (still before arming): just keep going — `scan.send(event)` buffers events and ships them automatically every 96 events, or `scan.learn(events)` sends one window immediately.

Step 2 — Arm

When teaching is done:

```
report = scan.arm(margin=1.75)
```

Arming analyses the fingerprint library, organises it into detection layers, and sets the alarm thresholds. The response is a calibration report. Two numbers worth writing down:

- **lock_threshold** — the tripwire distance. Anything measured further than this from normal raises a flag. In our validation runs this was around 0.09–0.11.
- **healthy_reference** — how far ordinary healthy behaviour typically sits from the library. Lower and clearly below the lock is good. In our validation runs this was around 0.05–0.06.

About margin: 1.75 is our validated recommendation — it captures the most attacks. Use 2.0 if your agent's behaviour is very regular and you want fewer false alarms.

Step 3 — Guard

Once armed, send live behaviour the same way you taught:

```
for event in live_events:
     scan.send(event)      # now ships guard windows automatically
```

Or check one window right now:

```
verdict = scan.guard(todays_events)
 print(verdict["verdict"])     # "MATCH" or "OUT_OF_DISTRIBUTION"
 print(verdict["distance"])    # how far from normal (compare to your lock)
```

Reading a verdict:

- `MATCH` — normal. Nothing to do.
- `OUT_OF_DISTRIBUTION` — unusual. The event is waiting in your review queue, and if you registered an alert webhook, you have been notified.
- `distance` vs your `lock_threshold` tells you how close the call was. A normal week in our validation runs measured around 0.10–0.11 against a lock of about 0.10; an injected attack measured about 0.118.

Step 4 — Review the queue

```
queue = scan.ood_list()
 for item in queue["events"]:
     print(item["ood_id"], item["label"], item["distance"])
```

For each waiting item, decide:

It is fine (new kind of normal work):

```
scan.label(item["ood_id"], label="approved_new_report", promote=True)
```

The unusual behaviour is moved into the library as new fingerprints, and BRAIN_SCAN re-arms itself automatically at the same margin. The same behaviour will now MATCH. One review teaches every fingerprint inside that window, not just the single most unusual one.

It is bad (attack, bug, misuse):

```
scan.label(item["ood_id"], label="prompt_injection", promote=False)
```

Nothing is taught. The event stays on record under your chosen name, and BRAIN_SCAN keeps catching it.

Step 5 — Live with it

- Review the queue regularly. Every review either grows the library (promote) or sharpens the threat record (label only).
- If your agent's job changes significantly, teach the new normal through the review loop rather than re-seeding from scratch.
- If you call `learn()` directly after arming, the agent drops back to teaching mode — you must call `arm()` again before guarding resumes. The review loop is the safer habit.

7. Alerts: get told when something happens

Pass an HTTPS webhook when you create the client (or on any learn/arm call):

```
scan = BrainScan(api_key="...", agent_id="support-bot",
                 alert_webhook="https://your-server.example/alerts")
```

BRAIN_SCAN posts an alert to that URL when behaviour stays unusual. Two built-in protections stop alert floods: an alert only fires after several unusual windows in a row (default 3), and repeat alerts wait a cooldown (default 15 minutes).

8. Debugging: help yourself first

Most problems have a known shape. Work through this section before contacting us — you will solve most issues in minutes.

8.1 First tool: read the whole response

Every response carries a `status` field and usually a `receipt` field. The receipt is a plain-English explanation written for you. Read it first — it usually names the exact problem.

Errors raise `BrainScanError` in the SDK. Print the whole thing:

```
from brainscan import BrainScanError
 try:
     scan.guard(events)
 except BrainScanError as e:
     print(e.status)     # the HTTP code
     print(e)            # the server's own explanation
```

8.2 Second tool: check the agent's state

```
print(scan.status())
```

You will see the agent's phase (`LEARNING` or `ARMED`) and how many fingerprints it holds. Half of all beginner issues are revealed right here — for example, guarding while still in `LEARNING`, or a library with far fewer fingerprints than expected.

8.3 "EVERYTHING is flagged unusual, even normal behaviour"

The most common issue, and almost always one of these:

1. **Thin first teaching call.** Your first learn pooled only one small window. Check: did your first learn response contain `BS-W2001`? Fix: delete the agent (Section 10), then re-teach with `learn_curriculum()` over a representative bundle.
2. **Wrong detection channel.** Check for `BS-W2002`. Fix: delete and re-teach with the default settings (the SDK's default channel, `feat_gram`, is the validated one).
3. **Forgotten `ignore_keys`.** Timestamps and request IDs make every window look unique. There is no way to add them after the first teach — delete, set `ignore_keys`, re-teach.
4. **Margin too tight for a noisy agent.** Re-arm with `margin=2.0`.

8.4 “NOTHING is ever flagged”

- Is the agent actually armed? Check `scan.status()`.
- Did you review-promote so broadly that the library now accepts everything? The queue history (`ood_list()`) shows what was promoted.
- As a last resort: delete, re-teach, re-arm at margin 1.75.

8.5 Common errors, decoded

What you see	What it means	What to do
`No API key...` (before any call)	The SDK has no key	Pass `api_key=` or set the env var
HTTP 403 SECURITY ALERT	Key unknown, inactive, or malformed	Check the key string for typos; ask us to confirm it is active
HTTP 400 `AGENT_NOT_ARMED`	You guarded before arming (or after a fresh learn/forget)	Arm first; re-arm if you taught again
HTTP 404 `AGENT_NOT_FOUND`	That agent name has no library on your key	Check spelling of `agent_id`; was it forgotten?
HTTP 400 `OOD_NOT_FOUND`	The review item ID is wrong or already handled	List the queue again with `ood_list()`
HTTP 429 SERVER_BUSY	The service is at capacity this second	Nothing — the SDK retries automatically
HTTP 500 with `BS-E...`	A fault on our side	Quote the code to us (Section 8.7)
pip: “does not appear to be a Python project”	The install URL did not end in the `.whl` filename	Use the exact command in Section 3
First call takes ~20 s	Cold start after quiet period	Normal; the SDK waits patiently

8.6 Warning codes (teach-time, in `protocol_warnings`)

Code	Meaning	Fix
BS-W2001	First teach pooled too few events (< 240) to build a good measuring stick	Delete the agent; re-teach with `learn_curriculum()` over a representative bundle
BS-W2002	JSON events arrived on the legacy `stats` channel	Delete the agent; re-teach with default settings (`feat_gram`)

Both are warnings, not errors — but act on them **before** arming.

8.7 When you do contact us

Include: the `BS-....` code (if any), your `agent_id`, the time it happened, and the response's `receipt` text. **Never send us your API key** — we never need it.

9. Housekeeping rules worth knowing

- **forgetting is soft**: `scan.forget()` moves the agent's library to a trash area. For 72 hours it can be restored by us; after that it is purged permanently.
- To protect against accidents and misuse, forget is rate-limited (10 calls per hour per key).
- Guard scans, teaches, and reviews are unlimited for normal use; the SDK automatically rides out brief busy periods.
- `ignore_keys`, `chunk_size`, and `feature_set` are fixed at the first teach and inherited forever after. Changing them means starting a new agent.
- `scan.reset()` exists for our operators only (admin keys). Customers always want `forget()`.

10. Removing an agent

```
scan.forget()
```

That is the whole act — calling the method IS your confirmation, so the SDK sends the server's typed-confirmation for you. The library vanishes from service immediately; guarding it afterwards returns `AGENT_NOT_ARMED`, and a second forget honestly reports `AGENT_NOT_FOUND`. If you change your mind within 72 hours, contact us to restore it.

11. Method reference

Call	What it does
`BrainScan(api_key, agent_id, ...)`	Connect to your agent's library
`send(event)`	Buffer one event; ships automatically every 96 (teach while learning, scan while guarding)
`flush()`	Ship the buffered events right now
`learn_curriculum(windows)`	**Seed**: pool many normal windows into ONE first teach (sets the measuring stick correctly)
`learn(events=None)`	Teach one window now (or ship the buffer)
`arm(margin=None)`	Finish teaching, set thresholds, start guarding. Recommended margin: 1.75
`guard(events=None)`	Scan one window now (or ship the buffer as a scan)
`status()`	Agent phase and fingerprint count
`ood_list(limit=50)`	Your review queue, newest first
`label(ood_id, label, promote=False)`	Decide on a queue item; `promote=True` approves it into the library and re-arms automatically
`forget()`	Soft-delete this agent (72-hour undo)
`reset()`	Admin keys only — permanent destruction

12. Settings reference

Setting	Default	Notes
`api_key`	env var	Required. Keep it secret.
`agent_id`	`"default"`	One library per name.
`endpoint`	production service	Override only for staging.
`feature_set`	`"feat_gram"`	The validated detection channel for JSON events. Pass `None` only for pure-numeric CSV telemetry. Fixed at first teach.
`chunk_size`	`24`	Events per fingerprint. Fixed at first teach.
`window_size`	`96`	Events per shipped window (4 fingerprints).
`ignore_keys`	none	Noisy fields to drop (timestamps, IDs). Fixed at first teach.
`alert_webhook`	none	HTTPS URL for unusual-behaviour alerts.
`timeout`	120 s	Covers cold starts.
`max_retries`	4	Automatic retries on busy/network errors.

13. A complete example session

This mirrors a real, verified run against the production service (numbers from our validation data; yours will differ):

```
from brainscan import BrainScan

scan = BrainScan(api_key="...", agent_id="payroll-bot",
                 ignore_keys=["timestamp", "request_id"])

# 1. TEACH — one pooled seed over 13 normal weeks (1,633 events)
 scan.learn_curriculum(thirteen_weeks)
 #    -> LEARN_COMPLETE, 69 fingerprints, no warnings

# 2. ARM
 scan.arm(margin=1.75)
 #    -> ARMED, 3 detection layers, lock_threshold ~0.107,
 #       healthy_reference ~0.061

# 3. GUARD
 scan.guard(week_14)   # -> MATCH, distance 0.111
 scan.guard(week_16)   # -> OUT_OF_DISTRIBUTION, distance 0.118
 #    (week 16 contained an injected attack)

# 4. REVIEW
 scan.ood_list()                                  # see what is waiting
 scan.label("OOD001", "approved_shift", promote=True)   # teach it
 scan.label("OOD002", "prompt_injection")               # name the threat

# 5. REMOVE (when the agent is retired)
 scan.forget()     # -> AGENT_FORGOTTEN, restorable for 72 hours
```