ALU GUBI CLI & Driver — Operator Reference Manual

Commands, REST API, PlankServices, Herd Computing, and Homebrew Install

Shyamal Chandra
Sapana Micro Systems
sapanamicrosoftware@gmail.com
Public site: https://shyamalschandra.github.io/alu-gubi-cli/
Charter: https://github.com/shyamalschandra/alu_gubi

July 2026 · RM-GUBI-2026-01 · companion to TR-GUBI-2026-01

Abstract

This reference manual is the operator-facing companion to the industrial technical report. It documents how to install the encrypted `alu-gubi-cli` and `alu-gubi-driver` binaries from Homebrew, run the REST control plane, drive PlankServices diagnostics, execute **herd computing** over a virtual population of 10^{11} people via compact RNG JSON tokens, and emit Knuth–Fisher–Yates `mathi-alu-gubi` lifestyle queues. **No source listings** are included; operators consume authenticated binaries and published documentation only.

Contents

1 Scope and documents 3

2 Install (binary only) 3

3 First-time bootstrap 3

4 alu-gubi-cli command groups 4
4.1 Real-time scheduler . . . 4

5 alu-gubi-driver command groups 4

6 Herd computing (virtual googolplex / infinity / finite) 5
6.1 Batch rounds (`herd`) . . . 5
6.2 Real-time stream (`herd-stream`) . . . 5
6.3 Stream-agnostic pipeline (`--pipe`) . . . 5
6.4 Against a live API (batch) . . . 6
6.5 Key flags . . . 6

7 mathi-alu-gubi event queue 6

8 PlankServices 6

9 REST API surface (control plane) 7

10 Environment variables (selected) 7

11 Homebrew supply chain 7

12 License 8

1 Scope and documents

Table 1: Public documentation map

Artifact	Anchor / path	Role
This manual (PDF)	`#reference-manual`	Operator command & API reference
Industrial report	`#paper`	Architecture & supply-chain threat model
HOWTO (Markdown)	`#howto` · `docs/HOWTO.md`	Day-one checklist
Citations	`CITATIONS.md`	Dual conditioning sources
License	`/license/`	CCSL-1.0a + NOTICE
Demo media	`#demo-media`	Pages screenshots + herd-stream movie

Citations (both required). Herd computing and PlankServices coordination are conditioned by (a) conversations with **Kailash Chandra** on *Introduction to Simulation* (Florida Institute of Technology, Space Coast / NASA Kennedy region), and (b) reading **Victor R. Lesser** online (UMass Amherst Multi-Agent Systems Lab: https://mas.cs.umass.edu/lesser.html). See `CITATIONS.md`.

2 Install (binary only)

Public distribution is macOS arm64 encrypted release (format v2 hybrid envelope: ML-KEM-768 key wrap, AES-256-GCM bulk, ML-DSA-65 signed manifest, homomorphic seal).

```
brew tap shyamalschandra/alu-gubi-cli
brew trust --formula shyamalschandra/alu-gubi-cli/alu-gubi-cli
brew install alu-gubi-cli
brew trust --formula shyamalschandra/alu-gubi-cli/alu-gubi-driver
brew install alu-gubi-driver
alu-gubi-cli --help
alu-gubi-driver --help
```

Tap repository: https://github.com/shyamalschandra/homebrew-alu-gubi-cli. After a release rebuild, refresh with `brew update && brew reinstall alu-gubi-driver`.

3 First-time bootstrap

1. `alu-gubi-cli ui` — stack dashboard
2. `alu-gubi-cli moby-up -d` — start Ollama via GubiMoby (no Docker Desktop)
3. `alu-gubi-cli ollama-pull` — pull `llama3.3` + `gemma3:1b`
4. `alu-gubi-cli migrate` — apply schema (in-memory unless `DATABASE_URL` set)
5. `alu-gubi-cli serve` — REST API on `http://127.0.0.1:8080`
6. (optional) `alu-gubi-driver live` — continuous emoji/unicode TUI against the API

One-shot field demo:

```
alu-gubi-cli automate --moby-up --pull-models \
  --zip 94102 --display-name "Demo Citizen"
```

4 alu-gubi-cli command groups

Commands use kebab-case. Representative groups:

Table 2: CLI command groups (server binary)

Group	Examples
Dashboard / brew	`ui`, `ui --watch`, `brew`
GubiMoby	`moby-up -d`, `moby-down`, `moby-ps`
Ollama	`ollama-pull`, generative + agentic tiers
Data	`migrate`, optional file-backed `DATABASE_URL`
API	`serve`, `serve --rt-profile hard-realtime`
RT offline	`rt-status --profile balanced`
PlankServices	`plank-status --pulse`
Domain demo	`automate`, coupons, points, CoL, Bindi

4.1 Real-time scheduler

Async deadline classes (`critical` / `interactive` / `background` / `idle`) with admission control and low-power idle policy. Example hard-realtime style launch:

```
export GUBI_RT_MAX_PARALLEL=4
export GUBI_RT_IDLE_SLEEP_MS=0
alu-gubi-cli serve --rt-profile hard-realtime --rt-max-parallel 4
```

Kernel-enforced hard RT requires an RTOS / PREEMPT_RT deployment; the CLI bounds application-level deadlines only.

5 alu-gubi-driver command groups

The driver is a thin pure-Rust HTTP client for `alu-gubi-cli serve`.

Table 3: Driver commands

Command	Purpose
`demo`	One-shot emoji/unicode panel tour
`live` / `watch` / `tui`	Continuous ratatui showcase
`health`	`GET /health`
`col-summary` / `col-zip`	Cost-of-living endpoints
`issue-coupons` / `award-points`	Domain POSTs
`agent-models`	Agentic model listing
`rt-status`	Scheduler status + contract
`plank-status`	Plank graph + rosout edge logs
`mathi-queue`	Knuth–Fisher–Yates lifestyle queue
`herd`	Herd computing → RNG JSON tokens
`get` / `post`	Raw REST helpers

Global options include `--base` (`GUBI_API_URL`, default `http://127.0.0.1:8080`), `--raw`, `--color` / `--no-color`, and `--vault-token`.

6 Herd computing (virtual googolplex / infinity / finite)

Herd computing covers a **virtual** population at one of three scales — never allocated in memory:

- `googolplex` — $10^{(10^{100})}$ (default; symbolic)
- `infinity` — unbounded fractional shard domain
- `finite` — legacy 10^{11} (100 billion)

Lightweight PlankServices generator shards emit compact Knuth LCG **RNG JSON tokens** with **opaque 16-byte person ids**. Batches are sealed into **Keystone-inspired enclave pages** (AES-256-GCM + homomorphic integrity seal) so external parties cannot modify sealed memory without detection.

```
alu-gubi-driver herd --scale googolplex --dry-run --raw
alu-gubi-driver herd --scale infinity --dry-run --raw
alu-gubi-driver herd-stream --scale googolplex --dry-run
```

6.1 Batch rounds (`herd`)

Finite emit→ingest rounds with an interval pause (scripting / report):

```
alu-gubi-driver herd --dry-run
alu-gubi-driver herd --dry-run -g 64 --batch-size 128 -r 8 --interval 10
```

6.2 Real-time stream (`herd-stream`)

Tick-driven continuous simulator with ratatui UX (not batch rounds). Micro-batches ride the stream clock (`--tick`); keys: `q` quit, space pause, `[/]` chunk, `-/+`/tick.

```
alu-gubi-driver herd-stream --dry-run
alu-gubi-driver herd-stream -g 32 --chunk 2 --tick 20 --dry-run
alu-gubi-driver herd-stream --no-tui --duration 3 --dry-run
# against live API:
alu-gubi-cli serve
alu-gubi-driver herd-stream --chunk 2 --tick 20
```

Aliases: `stream`, `herd-live`.

6.3 Stream-agnostic pipeline (`--pipe`)

Every source (emit, HTTP NDJSON/SSE, Plank bus) frames as one NDJSON `PipelineItem`. The same `--pipe` sinks apply to `herd-stream` and `herd-subscribe`:

```
stdout | tcp://HOST:PORT | unix:///path.sock | file:///path | plank-ingest
```

```
alu-gubi-driver herd-stream --dry-run --no-tui --pipe stdout --duration 5
alu-gubi-driver herd-stream --dry-run --no-tui --pipe file:///tmp/herd.ndjson
alu-gubi-driver herd-stream --pipe plank-ingest --scale googolplex
alu-gubi-driver herd-subscribe --format ndjson --pipe stdout
curl -N 'http://127.0.0.1:8080/v1/plank/herd/stream?format=ndjson'
curl -N 'http://127.0.0.1:8080/v1/plank/herd/stream?format=sse'
```

6.4 Against a live API (batch)

```
alu-gubi-cli serve   # terminal 1
alu-gubi-driver herd --raw
curl -s http://127.0.0.1:8080/v1/plank/herd/status | jq .
```

6.5 Key flags

Table 4: Herd simulation flags

Flag	Default	Meaning
`-g / --generators`	32	Lightweight shards covering N_{herd}
`--batch-size`	64	Tokens per generator per round
`-r / --rounds`	4	Emit→ingest rounds
`--interval`	25 ms	Pause between rounds (timely acks)
`--seed`	mathi default	Knuth LCG seed (`MATHI_SEED`)
`--temperature`	1.45	Stamped agentic sampling
`--top-k`	96	Stamped agentic top-k
`--dry-run`	off	Local emit only (no POST)
`--raw`	off	JSON instead of TUI panels

Aliases: `herd-sim`, `herd-compute`. Regenerate the website movie with `./scripts/render-demo-media.sh` (pure Rust `alu-gubi-demo-media`).

7 mathi-alu-gubi event queue

Knuth–Fisher–Yates shuffle of fake UBI payment and lifestyle events for the `mathi-alu-gubi` agentic client (high temperature / high top_k defaults).

```
alu-gubi-driver mathi-queue --raw
alu-gubi-driver mathi-queue --seed-mode operator --seed 42 -n 32 --raw
alu-gubi-driver mathi-queue --seed-mode entropy --priority --push --raw
```

Seed modes: `fixed` | `operator` | `entropy`. Environment: `MATHI_TEMPERATURE=1.45`, `MATHI_TOP_K=96`. Aliases: `mathi`, `generate-queue`.

8 PlankServices

Invented by Shyamal Chandra. Modular ROS-inspired pub/sub inside the encrypted binary.

- **Nodes:** api, rt, col, coupons, points, bindi, herd-bridge, herd-gen-*, plank-logger, viz
- **Topics:** `/gubi/*`, `/plank/rosout`, `/plank/power`, `/plank/herd/tokens`
- **Edges:** each hop logs level, stamp, seq, latency µs, estimated mW
- **RT class:** per-topic deadline class (critical / interactive / background / idle)

```
alu-gubi-cli plank-status --pulse
alu-gubi-driver plank-status --pulse
curl -s http://127.0.0.1:8080/v1/plank/status | jq .
curl -s http://127.0.0.1:8080/v1/plank/rosout | jq .
```

9 REST API surface (control plane)

Default base URL: `http://127.0.0.1:8080`. Representative routes:

Table 5: Selected HTTP routes

Method	Path	Notes
GET	`/health`	Liveness
GET	`/v1/col/summary`	CoL rollup
GET	`/v1/col/zip/{zip}`	Per-ZIP CoL
POST	`/v1/coupons/issue`	Coupon issuance
POST	`/v1/points/award`	Points award
GET	`/v1/agents/models`	Model inventory
GET	`/v1/rt/status`	Scheduler telemetry
GET	`/v1/rt/contract`	Deadline contract
GET	`/v1/plank/status`	Plank graph
GET	`/v1/plank/rosout`	Edge rosout buffer
POST	`/v1/plank/herd/ingest`	Token batch ingest
GET	`/v1/plank/herd/status`	Herd status JSON

Vault-compatible KV paths under `/v1/secret/data/*` use the process GubiVault (default dev token `alu-gubi-dev-token` via `VAULT_TOKEN`).

10 Environment variables (selected)

Table 6: Common environment variables

Variable	Role
`GUBI_API_URL`	Driver base URL
`DATABASE_URL`	Optional file-backed SQLite
`VAULT_TOKEN` / `VAULT_MASTER_KEY`	Secret plane
`LIVE_INTEGRATIONS`	Enable live Census/BLS-style feeds
`GUBI_RT_*`	Parallelism, idle sleep, storage batch
`MATHI_SEED`	Knuth LCG seed for mathi/herd
`MATHI_TEMPERATURE` / `MATHI_TOP_K`	Agentic sampling stamps
`NO_COLOR`	Disable ANSI color

11 Homebrew supply chain

1. Encrypted tarballs hosted on GitHub Pages under `releases/`.
2. `releases/manifest.json` publishes SHA-256 digests.
3. Formulas in tap `shyamalschandra/alu-gubi-cli` pin those digests.
4. Operators must `brew trust` before install (Homebrew 6+ tap trust).

Publish pipeline (maintainers): `./scripts/update-all.sh` compiles docs PDFs, runs tests, builds the site, pushes `master`, and syncs the Homebrew tap. Encrypted binary rebuilds occur on crate changes or `workflow_dispatch` of the GitHub Pages workflow.

12 License

CCSL-1.0a (Chandra Credit Software License). See https://shyamalschandra.github.io/alu-gubi-cli/license/ and `NOTICE.md`. Conditioned in part by the OpenAI ChatGPT note linked from the public site.

Document control

- Identifier: RM-GUBI-2026-01
- Companion report: TR-GUBI-2026-01 (`docs/alu-gubi-cli.pdf`)
- Source: `docs/reference_manual.tex`
- Public PDF: https://shyamalschandra.github.io/alu-gubi-cli/docs/reference_manual.pdf
- Embedded preview: https://shyamalschandra.github.io/alu-gubi-cli/#reference-manual