ChatGPT

Spectra CLI – Unified Kobalt Σ Model Sandbox Interface

Spectra CLI is a comprehensive Unix command-line interface designed to expose the full capabilities of the Arquolab IO Sandbox models within the Spectra Gallery framework. It leverages the **Kobalt Σ (Sigma) model** architecture to bring together swarm intelligence, ecological simulations, AI agents, and archetypal semantic frameworks into a single toolkit. The goal is to enable artists, researchers, and developers to run and visualize complex generative simulations and data projections in real-time from the command line, with support across multiple programming environments.

Key capabilities of Spectra CLI include:

- **Swarm Intelligence Simulation:** Real-time boid swarm simulations (Sigma Algea Swarm) with 3D visualization and live status feedback.
- **Predator–Prey Ecosystem Modeling:** Lotka-Volterra predator/prey population dynamics with tunable parameters and output for analysis.
- **Sentient AI Agents:** Evolving agents (e.g. Alpha Evolve) with persistent neural network state, demonstrating self-adaptive "sentient" behavior.
- **Dystopian Interactive Scenarios:** Experimental simulations (e.g. *Quantum Jellyfish*, *HyperGuardian*) that incorporate chaotic or multi-sensor inputs for cyberpunk/dystopian art experiences.
- **Agentic Self-Improvement Systems:** Autonomous agents that mutate and improve over generations, with secure state persistence and encryption.
- **Archetypal Typology Forms:** Dynamic form interfaces based on Kobalt schemas, allowing users to select high-level themes (activism, humanism, etc.) that inform simulation parameters.
- **Cross-Language Integration:** Operable from Bash and able to interface with Node.js, Python, Go, Pascal, and Fortran components, using standard data formats (JSON, CSV) for interoperability.
- **Dashboard & Data Visualization:** An interactive dashboard (web or TUI) that layers control panels, forms, and visual output (2D/3D graphs) in real-time, without information loss (non-entropic visualization).

Architecture Overview (Kobalt Σ Model)

Spectra CLI is built on the **Kobalt Σ model architecture**, which blends multi-agent systems with semantic-driven interfaces. The Arquolab Sandbox was originally conceived as a hackable, open-source playground for generative art, data visualization, science, and interaction design [1]. Spectra CLI inherits this philosophy, organizing functionality into modular *agents* and *services* that correspond to conceptual archetypes (Alpha, Sigma, etc.) within the sandbox. Each module operates in a layered architecture to ensure real-time performance and persistent state (preventing entropy or data loss between runs).

Some core components of the architecture include:

- **AlphaEvolveAgent (Sentient AI):** An agent that simulates a self-evolving entity, storing its state (e.g. rotation rates, neural network weights, generation count) in a database for continuity [2].

This represents the “sentient” archetype – the CLI can start, stop, and query this agent to observe an AI learning or mutating over time.

- **SigmaAlgeaSwarm (Swarm Simulation):** A swarm intelligence module that manages a flock of boids (bird/agent models) with positions and velocities persisted in state storage [3]. A particle-tracking sub-system analyzes boid motion and feeds data into the state, ensuring the swarm’s behavior can be tracked and resumed (the basis of the “σ” model). This agent corresponds to the *swarm* archetype and powers flocking simulations.
- **SelfImprovingAgent (Agentic Evolution):** An autonomous agent which evolves a lightweight neural network through mutation and evaluation cycles [4]. It can encrypt its parameters via a cipher module before storing results, demonstrating an **agentic** self-improvement loop. The CLI can employ this for scenarios where an AI tries to optimize or learn over generations (e.g. genetic algorithms).
- **QuantumJellyfish & HyperGuardian (Experimental):** Additional agents extend the sandbox into complex, interactive realms. For example, *HyperGuardianAgent* drives the **hyperbolic universe** demo, using a neural network influenced by live audio and sensor input to produce evolving 3D visuals [5]. These modules exemplify “dystopian” or futuristic scenarios – mixing chaotic inputs (audio, orientation) to generate unpredictable art. The CLI architecture treats these as pluggable modules that can be launched or halted similarly to other agents [6] [7].
- **Kobalt Schema & Forms:** The Kobalt component of the architecture handles semantic content and user input schemas. A PDF-derived schema of conceptual topics is converted to JSON for dynamic form generation [8]. This means the CLI can present forms or prompts for selecting high-level **archetypal typologies** (e.g. “Alpha – Manifesto”, “Beta – Activism”, “Humanism”, “Free Will”, etc.) [9]. These choices can influence the behavior or configuration of simulations (for instance, choosing *Activism* or *Autonomy* might load a particular preset or data set). By formalizing such inputs, the system infuses **semantic context** into the numeric simulations.

Behind the scenes, these components interact through layered services to maintain a **non-entropic** stateful system. For example, the SigmaAlgeaSwarm uses a ParticleTracker to continually save boid coordinates to a `SigmaSwarmState` store [3], and the AlphaEvolveAgent writes its evolving parameters to a MongoDB collection for persistence [10]. A cipher module can secure sensitive parameters (e.g. the SelfImprovingAgent’s learned data) before storage [11]. This layered design (agents → trackers → database → cipher) ensures that data is not lost between sessions and can be built upon cumulatively. In essence, the CLI’s architecture ties together real-time simulation engines with persistent data models and semantic overlays (themes/forms), following the Kobalt Σ model principle of **stacking multiple layers** (from raw physics to high-level meaning).

Swarm Intelligence Simulation Module (Sigma)

One of the showcase features of Spectra CLI is the swarm simulation module, corresponding to the Sigma archetype. This module simulates flocking behavior (often illustrated with “boids”) in real time. Users can initiate a swarm simulation via the CLI, which will spawn the SigmaAlgeaSwarm agent. The simulation parameters – number of agents (boids), alignment/cohesion/separation weights, speed limits, neighbor radius, etc. – are fully configurable. For instance, the CLI can expose options analogous to the web form sliders (boid count, alignment, cohesion, separation strengths) [12] so that the user may tune swarm behavior before starting the run.

Once launched (e.g. with a command like `spectra-cli swarm start --boids 50 --alignment 1.0 ...`), the swarm runs continuously and its state is updated and logged. The system can report status information such as whether the swarm is running and how many boids are active, which is available through status commands or the dashboard [13]. Internally, a fetch-loop polls the swarm’s state at short intervals (on the order of 1 second) [14], enabling live updates to any connected interface.

In a visualization context, the positions of boids are rendered (e.g. as spheres in a 3D scene) and updated each frame to reflect their movement [15]. The persistent design means the swarm's state (positions, velocities) can also be saved or analyzed even after the simulation stops [3]. This Swarm Intelligence module demonstrates emergent behavior and is useful for studying alignment and clustering phenomena, or simply creating generative art based on flocking patterns.

Predator–Prey Ecosystem Simulation

Spectra CLI also includes a classic **predator–prey** simulation module, showcasing ecological dynamics via the Lotka–Volterra equations. This module models two populations (e.g. rabbits vs. wolves) and how they affect each other's growth. Via the CLI, a user can run the predator-prey simulation for a specified number of time steps (e.g. `spectra-cli eco predator-prey --steps 1000`). The underlying model computes population changes in discrete time using differential equations for growth and interaction. For example, given initial populations and rate constants, at each step prey increase by a factor of `a*prey` and are eaten at rate `b*prey*predator`, while predators increase by `d*prey*predator` and die off at rate `c*predator` [16]. These equations produce cyclical oscillations in population sizes, emulating realistic ecosystem booms and busts.

The CLI's implementation of this prints the results in a structured format (CSV-like output with columns for step, prey count, and predator count) [17], allowing easy redirection to files or further analysis. For instance, the first line of output is a header (`step,prey,predator`) followed by lines of numeric values per time step [17]. This design facilitates integration with data science tools: one could pipe the output to a file and plot it using Python or R to visualize the predator-prey cycle. In the Spectra CLI's interactive dashboard mode, it's envisioned that the population curves can be plotted in real-time as the simulation runs, giving a visual graph of prey and predator populations over time. Users can adjust parameters (birth rate, predation rate, etc.) through CLI options or form inputs before running, to explore different scenarios (e.g. a more "dystopian" scenario where predators overwhelm prey leading to collapse, or a balanced scenario of coexistence). This module brings a science-lab style **data projection** aspect – results can be projected onto charts or further processed statistically, embodying a *Science Data Lab* approach to simulation output.

Sentient AI Agent Module

The **sentient agent** functionality of Spectra CLI is represented by agents like the *AlphaEvolveAgent*. This module allows users to experiment with an AI-like entity that "lives" and learns within the sandbox. When activated (via `spectra-cli agent start alpha` or similar command), the Alpha agent begins to iterate an internal state update loop. It modifies parameters (for example, a set of neural network weights or a rotation value) slightly on each generation and evaluates them – mimicking a rudimentary learning process [18] [19]. The agent's state is persistently stored so that runs can be stopped and resumed without losing progress [20]. Key metrics like the current generation count, the agent's internal "energy" or fitness, and whether it's actively running are accessible through status queries or displayed on the dashboard.

What makes this agent "sentient" in context is its autonomous, continuous adaptation and state retention. The CLI enables monitoring of this process: for example, one can issue `spectra-cli agent status alpha` to retrieve the latest state (including the generation number and any learned parameters) as JSON. Because the architecture stores these parameters in a MongoDB collection (e.g. `AlphaAgentState`) [2], the agent's knowledge is cumulative (non-entropic). In practice, this could be used to evolve simple behaviors or patterns over time. Although the current Alpha agent is a simple prototype (updating a dummy rotation rate and random weights), it establishes

the pattern for more sophisticated **sentient AI simulations**. Future extensions might plug in a real neural network or evolutionary strategy for the agent to solve a problem or generate art. The Spectra CLI provides the controls to start/stop these agents and observe their evolution in conjunction with other modules – for instance, the output of a sentient agent could be used to influence a visual (giving an artwork that "learns" each time it runs).

Dystopian & Complex Scenario Modules

Beyond traditional simulations, Spectra CLI supports more **experimental and immersive scenarios** often described as *dystopian* or futuristic art pieces. These scenarios combine multiple inputs and modalities to create complex behaviors. A prime example is the **HyperGuardian** module, which powers an interactive "hyperbolic universe" demonstration in the Spectra ecosystem [21]. In this scenario, a neural-network-driven agent is connected to external sensory inputs: real-time audio from the microphone and device orientation sensors (e.g. accelerometer/gyroscope data). When the HyperGuardian agent is running, each animation frame it reads the current audio frequency spectrum and device tilt, and feeds those along with some randomness into its neural network [5]. The outputs of the network continuously adjust a 3D scene – for instance, deforming or recoloring a guardian figure in a hyperbolic 3D space. The result is an ever-evolving visual that reacts to the environment and user presence, evoking a cyberpunk aesthetic.

Using Spectra CLI, users can launch such a scenario (e.g. `spectra-cli demo hyper-guardian start`) which would initialize the necessary processes: accessing sensors (if on a device), starting the agent's update loop, and opening the visualization. This represents a **layered, real-time simulation** where multiple data streams (sound, motion, random noise) layer together to drive the output. The architecture's capacity to handle this in real-time without losing data (non-entropic) means the agent retains an internal state (such as an energy level and generation count) that evolves with the input [22]. If the simulation is paused or resumed, the state persists, making the experience continuous.

Another experimental module referred to in the system is the *QuantumJellyfishAgent*, hinting at a simulation with chaotic or unpredictable elements. While not visualized in the same way, one can imagine it as a generative algorithm producing complex patterns (perhaps "swarming" like jellyfish in an abstract space). The CLI provides endpoints to start and stop this agent as well [6], suggesting it can be run headlessly or with a custom visualization. These dystopian or avant-garde modules highlight the flexibility of Spectra CLI – it is not limited to textbook simulations, but can serve as a platform for interactive art installations or experiential tech demos. By exposing them under the same CLI umbrella, creators can script these experiences, incorporate them into pipelines, or run them on different environments (for example, driving a physical installation via a headless CLI controlling the HyperGuardian agent).

Agentic Self-Improvement and Secure Persistence

The notion of **agency** and self-improvement is woven throughout Spectra CLI's design. In particular, the SelfImprovingAgent module provides a template for how an autonomous system can iterate on itself. When invoked (possibly via `spectra-cli agent start self-improving`), this agent will attempt to optimize a small neural network or rule set by repeatedly mutating it and measuring some outcome [4]. Although the specifics of the evaluation criterion can be defined by the user or scenario, the CLI ensures that after each cycle, important parameters can be stored (for example, writing to a `Tribe` or `Axiome` collection as noted in the docs [11]). Uniquely, the system can employ a **Cipher module** to encrypt these parameters before storage, adding a layer of security or obfuscation to the agent's knowledge [11]. This could be important if the evolution involves sensitive data or simply as an artistic

choice (e.g. embedding hidden messages or using blockchain-like transparency where only hashed values are stored).

From the user's perspective, Spectra CLI exposes this agentic process in a controllable way. One might configure how many generations to run or what mutation rate to use through CLI flags, then observe progress via periodic status outputs or a log. Because the SelfImprovingAgent is designed to run indefinitely until stopped, the CLI supports stopping criteria or manual halt (`spectra-cli agent stop self-improving`). Thanks to persistent storage, a halted run can be resumed later, continuing from the last best state instead of starting over – reflecting the **non-entropic** design that accumulates improvements. This capability is especially useful for long-running evolutionary art or machine learning experiments; the CLI essentially provides a headless training loop manager with the ability to plug in visualizations or data exports as needed. In summary, the agentic self-improvement module in Spectra CLI empowers users to orchestrate self-learning algorithms from the command line, with built-in features for data integrity and security that align with the Spectra ethos of open-source yet privacy-conscious development.

Archetypal Typology & Semantic Configuration

A distinguishing feature of Spectra CLI is its integration of **archetypal typology semantics** – meaning users can configure or influence simulations using high-level conceptual categories. This is enabled by the Kobalt schema system. The CLI can present dynamic forms or accept parameters that correspond not just to numeric values, but to semantic choices. For example, a user might select a theme like "Avant-Garde vs. Against Radicalisation" or "Community vs. Anarchism" as part of setting up a simulation scenario. These options come from a curated list of topics extracted from the Kobalt framework. In fact, the project's dummy Kobalt schema enumerates topics ranging from *Equality and inclusion* to *Moral responsibility*, and coded archetypes like **Alpha (Manifesto)**, **Beta (Activism)**, etc., up through ideals like *HUMANISM*, *FREE WILL*, *PRIVACY*, *AUTONOMY* [9].

Within Spectra CLI, these choices can map to predefined configurations or data sets. For instance, choosing an **Alpha** archetype might load a "manifesto" preset for an AI agent (perhaps aggressive learning toward a goal), whereas **Beta (Activism)** might configure the system for community-oriented behavior (e.g. swarm agents cooperating more strongly). Similarly, selecting *Privacy–Transparency* or *Freedom–Autonomy* could toggle how data is logged or how much randomness vs. control is in a simulation. The CLI ensures that such semantic selections are not just abstract labels: thanks to the JSON schema approach, each choice corresponds to a value that the underlying code understands and can act upon [23] [24]. The CLI might provide a command like `spectra-cli config --topic "Avant-Garde | Discovery"` which updates a global or scenario-specific setting. In an interactive mode, the CLI's dashboard could render a dropdown or checklist of these archetypal themes (driven by the Kobalt JSON schema) for the user to pick from.

By incorporating archetypal typologies, Spectra CLI bridges the gap between raw technical parameters and **human-centric concepts**. This means simulations can be configured in terms of narratives or philosophies, not just numbers. It adds an educational and exploratory dimension: users can see how different guiding principles (e.g. a scenario emphasizing *Humanism* vs. one emphasizing *Anarchism*) might lead to different outcomes in the simulation. Under the hood, the framework might map these to algorithmic differences – for example, a *Community* archetype could cause agents to favor cooperative behavior, whereas an *Alpha* archetype might introduce a dominant leader agent, etc. The CLI thus serves not only as a tool for computation but as a canvas for storytelling and hypothesis testing, where **typology semantics** shape the experience. This design is aligned with the creative, multidisciplinary spirit of the Spectra project, which spans art, science, and social themes [25].

Cross-Language and Cross-Platform Support

Although Spectra CLI is conceptually centered on a Unix shell interface, it is built with **multi-language interoperability** in mind. The underlying sandbox models are primarily implemented in JavaScript/Node.js (as inherited from Arquolab), but the CLI framework does not confine users to Node. Instead, it acts as a unifying layer that can invoke or communicate with components in various languages:

- **Bash/Shell:** At its core, Spectra CLI is a command-line tool accessible from any Unix shell. Users can incorporate it into shell scripts, pipelines, or cron jobs. All outputs (status, data streams) are printed to standard output or files in plain text or structured formats, making it easy to pipe results into other shell utilities.
- **Node.js:** Many simulation engines (swarm, agents, server endpoints) are implemented in Node/JavaScript within the Arquolab sandbox [26]. The CLI directly utilizes these – for example, running the Node scripts for predator-prey or starting the Express server for the dashboard. Because Node.js is a first-class environment here, the CLI can load modules or spawn Node processes as needed. This also means developers can extend the CLI by writing new modules in JavaScript and adding them to the CLI command set.
- **Python:** For data analysis and scientific computing tasks, the CLI can interoperate with Python. One approach is outputting data in CSV/JSON that Python scripts can easily consume (e.g. a user might do `spectra-cli eco predator-prey --steps 500 > out.csv` and then analyze that in pandas). Additionally, the design foresees the possibility of invoking Python directly for certain tasks – for instance, generating a Matplotlib plot of simulation results or using a Python machine learning library to guide an agent. The CLI could have subcommands that internally call Python scripts or Jupyter notebooks. By adopting **standard data schemas (JSON, CSV)** and web protocols for the dashboard, Spectra CLI ensures Python and other languages can hook in without friction [1].
- **Go:** Go can be used to compile critical components of Spectra CLI for performance and portability. For example, if packaging the CLI as a single binary is desired, a Go-based wrapper could embed Node or call into the JavaScript engines via APIs. Go's strength with concurrency might also be used for managing multiple simulation threads or handling real-time data streaming to the dashboard. The Spectra CLI design remains open to implementing certain compute-intensive modules (like a physics simulation or a complex optimization routine) in Go and linking it in. This provides a pathway to high-performance execution when needed.
- **Pascal & Fortran:** Recognizing that some scientific models and legacy algorithms are best expressed in older languages, Spectra CLI does not shy away from integration with Pascal or Fortran code. For instance, a highly optimized Fortran routine for solving differential equations or a Pascal implementation of a chaotic system could be compiled as a library and invoked by the CLI's Node or Go components. Another approach is using WebAssembly: existing Pascal/Fortran code can be transpiled to WASM or linked via C interfaces, then loaded into the Node.js environment. The CLI's architecture (with its emphasis on standard data exchange) means that as long as these components can read inputs and produce outputs in an expected format, they can be part of the toolchain. In practice, a user might never know what language a particular sub-module is written in – e.g., running `spectra-cli simulate three-body` could under the hood call a Fortran library for orbital calculations, but the CLI abstracts that detail. The inclusion of Pascal and Fortran underscores the **extensibility** of Spectra CLI: it can serve as a wrapper for both cutting-edge and classical code, uniting them under a common interface.

Importantly, all these language integrations are coordinated through a unified command syntax and data format conventions. The CLI outputs human-readable results by default, but can also output machine-readable data (e.g. JSON) when `--json` flag is used, making it easy for any language environment to parse results. Configuration files for the CLI (if any) could also be in a language-neutral

format (YAML or JSON) so that different ecosystems can generate or modify them. By designing Spectra CLI in a polyglot-friendly way, the tool becomes a **bridge between environments** – for example, a researcher could run a simulation from a Jupyter notebook by calling the CLI via a shell command, then load the results back into Python for analysis, or a web developer could call the CLI on a server to perform a heavy computation and return the result to a web client.

In summary, Spectra CLI's multi-language support ensures that the sandbox's rich functionality is accessible wherever it's needed: in shell pipelines, embedded in other applications, or as part of scientific workflows. It embraces the reality that creative coding and scientific computing often require picking the right language for the task, and provides the glue to make them work together seamlessly.

Interactive Dashboard and Visualization Toolkit

While command-line outputs and data files are useful, Spectra CLI also features an **interactive dashboard** that can be launched to provide a richer, real-time visualization and control experience. This dashboard is a web-based UI (served by an internal Express/Next.js application) that complements the CLI, effectively turning simulations into live visual experiments. When a user runs `spectra-cli dashboard` (or when certain simulations are started with a flag to open the UI), the system starts the Next.js *visualizer station* and opens a local webpage in the browser.

Within the dashboard, the interface is organized into panels and controls, mirroring the CLI's modules:

- **Toolbox & Module Selector:** A sidebar or top menu lists the available modules (Swarm, Predator-Prey, Agents, etc.). The user can select which simulation or agent to focus on. This acts as a toolbox of experiments, making it easy to switch context or run multiple modules concurrently. For example, one can start the Swarm and Predator-Prey simulations in parallel and toggle between their views.
- **Parameter Forms:** For the selected module, the dashboard presents input controls (forms, sliders, fields) to adjust parameters on the fly. These forms are generated from the same JSON schemas that the CLI uses. In the swarm module, the form includes fields for **Boids count, Alignment weight, Cohesion weight, Separation distance,** etc., as seen in the React component [12]. In the UI, these appear as sliders or number inputs. The user can tweak values and submit, upon which the CLI's backend API is called to apply the new parameters or restart the simulation with the updated settings [27]. This dynamic form-driven approach means that any new parameter or even new module added to the CLI can automatically produce a UI control, maintaining consistency.
- **Real-Time Visualizations:** The core of the dashboard is the live visualization panel. For swarm simulations, this is a 3D canvas (powered by Three.js) showing moving particles/boids in real time [15]. For predator-prey, it could be a 2D chart plotting population curves that update every few seconds. For agents, it might be gauges or textual readouts of their internal state (or even creative visuals if the agent is tied to one). The visualizer continuously pulls data from the CLI's backend – for instance, the swarm scene calls an API endpoint every second to get the latest boid positions [14], and updates the scene accordingly. The predator-prey chart could receive streaming data points. This **real-time feed** is crucial for an immersive experience, turning the CLI into a living dashboard rather than a static output.
- **Status and Logs:** Another panel or overlay shows status information and logs. This might include something like the *Swarm Status* readout (e.g. "Running: yes, Boids: 50") which is updated periodically [13]. Logs could show textual info such as each generation of an AI agent or important events (e.g. "Agent reached generation 100, increased performance by 10%"). This gives transparency into what the simulations are doing under the hood.

- **Layered Overlays:** The term *layered and non-entropic* in the context of visualization implies that multiple data layers can be shown together without overwriting or losing earlier information. In the dashboard, users can overlay different runs or aspects. For example, one could pause a simulation and retain its last state plotted on a graph, then run a new simulation and compare the outcome on the same chart (layering two data series). Or in the 3D view, the path history of boids might be drawn as trails rather than only showing current positions, so the motion traces accumulate (illustrating non-entropic retention of history). The interface might allow toggling these layers on/off. Thanks to persistent data capture in the backend, such layering is feasible – the CLI can retrieve full histories from the database or memory for visualization.

The **Science Data Lab projection** aspect comes into play via this dashboard. Users are effectively in a lab-like environment: they can manipulate initial conditions, watch experiments run in real-time, and see data projected into visual forms instantly. For example, in a predator-prey scenario, the dashboard could project the phase space (predator vs prey population plot) or a timeline graph of populations. In a swarm scenario, aside from the spatial view, it might project aggregate metrics (like average velocity or clustering coefficient of the swarm over time). By providing both control and insight, the Spectra CLI dashboard helps users iterate rapidly – much like a scientist tweaking an experiment and observing results on the lab instruments.

Technically, this dashboard is built using modern web frameworks and is integrated with the CLI's backend via a set of API endpoints [28] [29]. The design ensures that even if the dashboard is not used, the CLI by itself is fully functional (all actions are exposed via CLI commands and outputs). But when needed, the dashboard can be spun up to augment the experience. This two-mode approach (CLI-only vs CLI+UI) caters to both scripting usage and interactive exploration. Ultimately, the inclusion of a real-time, layered visualization interface transforms Spectra CLI from a mere collection of scripts into a powerful sandbox **workbench** – one where code, data, and visuals converge.

Usage Examples

To illustrate how one might use Spectra CLI in practice, consider the following scenarios:

- **Running a Basic Simulation:** A user wants to simulate a predator-prey system for 1,000 steps and visualize the results. Using Spectra CLI, they could execute:

  ```
  spectra-cli eco predator-prey --steps 1000 --output data.csv
  ```

  This runs the simulation with default parameters and saves the output to `data.csv`. Next, to quickly visualize it, they might use the CLI's built-in plotting (if available) or open the dashboard:

  ```
  spectra-cli visualize data.csv
  ```

  This could generate a pop-up graph or launch a small interface showing the prey and predator population curves over time. Alternatively, they could directly open the dashboard for a live run:

  ```
  spectra-cli dashboard --module predator-prey --steps 1000
  ```

  Which would launch the web UI, run the simulation, and show the graph updating in real-time.

- **Interactive Swarm Tuning:** Another user is exploring flocking behavior. They run:

```
spectra-cli swarm start --boids 20 --alignment 1.2 --cohesion 1.0 --
separation 1.5
```

The CLI starts the Sigma swarm with 20 boids and the given weight parameters. The user then opens the dashboard (the CLI might output the local URL or open it automatically) to see the 3D swarm. On the dashboard, they use sliders to increase "Boids" to 50 and decrease "Alignment" to 0.8, then hit **Update**. The CLI's backend receives these new parameters [30] and applies them, resulting in the visualization adjusting to a denser, less aligned flock. They observe the swarm forming tighter clusters (higher separation weight) and note the boid count update in the status panel. Satisfied, they stop the swarm with a command or button (`spectra-cli swarm stop`), and perhaps save the final state for later (`spectra-cli swarm export state.json`).

- **Launching a Sentient Agent Demo:** To demonstrate the system's unique AI capabilities, a user tries the HyperGuardian demo via CLI. They plug in a microphone and run:

```
spectra-cli demo hyper-guardian --duration 120
```

This command might start the HyperGuardianAgent for 120 seconds. The CLI will automatically launch the visualizer at `http://localhost:3000/hyperbolic` (as noted in the docs) [31]. Upon visiting that page, the user grants microphone permission. They see a glowing 3D icosahedron shape that pulses and changes color. As they play music or speak, the shape reacts – audio frequencies altering its energy and form. If they tilt a mobile device, the shape rotates accordingly [5]. During this time, the CLI terminal might show log messages like "Energy=0.82, Generation=42" as in the status JSON [32], indicating the agent's internal state changes. After 2 minutes, the CLI stops the demo and perhaps prints a summary of the agent's final energy level and generations run. This example shows how Spectra CLI can drive an interactive, sensor-rich experiment hands-free from the command line.

- **Batch Experiment with Archetypal Settings:** A researcher is interested in how different philosophical orientations impact a simulation's outcome. They decide to use the archetypal configuration feature to run multiple trials of the predator-prey simulation with slight rule variations corresponding to archetypes. Using Spectra CLI's semantic options, they run:

```
spectra-cli eco predator-prey --steps 500 --theme "Community" --output
comm.csv
spectra-cli eco predator-prey --steps 500 --theme "Free Will" --output
freewill.csv
spectra-cli eco predator-prey --steps 500 --theme "Anarchism" --output
anarchism.csv
```

Each run might internally adjust certain parameters (for example, "Community" could reduce the predation rate to model cooperative prey behavior, while "Anarchism" might remove any stabilizing factor leading to chaotic swings). After collecting the outputs, the researcher compares the `*.csv` files. They find that under **Community**, the populations reach an equilibrium, whereas **Anarchism** leads to more extreme oscillations. These high-level theme

inputs are drawn from the Kobalt schema of topics [33], illustrating how easy Spectra CLI makes it to incorporate and test abstract ideas. The entire process is automated via shell scripting, demonstrating the CLI's power in batch experimentation.

These examples underscore the versatility of Spectra CLI: whether used interactively or in automated pipelines, it enables a wide range of use cases from artful demos to scientific analysis, all using a coherent set of commands and interfaces.

Conclusion and Next Steps

The **Spectra CLI** brings together the rich functionalities of the Arquolab sandbox models and the Spectra Gallery ecosystem into a single, unified tool. By implementing the Kobalt Σ model semantics—encompassing swarm intelligence, predator-prey dynamics, sentient and agentic AI, and archetypal human-centered configurations—it transcends the typical boundaries of a command-line tool. Developers and creators can now seamlessly switch between coding, visualization, and conceptual exploration without leaving the CLI environment.

This document serves as a blueprint for the new repository (provisionally named `spectra-cli`) which will house the CLI implementation. Moving forward, the integration will involve refactoring or packaging the existing sandbox code into a library that the CLI commands can invoke. Key tasks include wrapping the Express server and Next.js visualizer to be launched on demand by the CLI, ensuring the multi-language hooks (e.g. preparing any Python or Fortran components) are in place, and thorough testing of each module's start/stop and data output functionality. Documentation will be generated alongside (in Markdown form, as exemplified here) to guide users in installing and using the CLI on various platforms.

By adhering to the design principles outlined above—**real-time interaction**, **layered persistent architecture**, **semantic richness**, and **cross-environment compatibility**—Spectra CLI is poised to become a powerful interface for generative art and scientific exploration. It transforms the Arquolab sandbox from a collection of disparate experiments into an integrated **command-line playground**. Users will be empowered to craft new simulations, combine modules in novel ways, and extract insights, all while tapping into the creative ethos of Spectra Gallery. The fusion of art, science, and open-source hackability in Spectra CLI exemplifies the project's innovative spirit, and sets the stage for community-driven growth as others contribute new modules or improvements (thanks to the open, modular architecture).

With Spectra CLI, what once required separate tools and environments can now be done in one place: run a swarm, evolve an agent, project data in visual form, tweak an ethos slider, and iterate – all in a few commands. This unified interface will not only save time but also inspire users to play with parameters and ideas, fostering **emergent creativity** at the intersection of code and concept. The successful implementation of this CLI will mark a significant milestone for the Spectra Gallery project, turning its theoretical "swarm pred-prey sentient dystopian agentic archetypal" semantics into a tangible, usable system that anyone can experiment with.

Sources:

- Arquolab IO Sandbox Whitepaper – project overview and key features [25] [1]
- Arquolab Sandbox Agent Architecture – design of Alpha, Sigma, and SelfImproving agents [2] [3] [4]

- Arquolab Sandbox Code – examples of simulation implementations and UI components (predator-prey model [17], swarm form parameters [12], swarm status API [14], hyper-guardian demo details [5], etc.)
- Kobalt Schema Data – archetypal topic list for dynamic forms [9] and JSON schema generation [8].

---

[1] [25] whitepaper.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/whitepaper.md

[2] [3] [4] [10] [11] [20] agent-architecture.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/agent-architecture.md

[5] [21] [22] [32] hyper-guardian-agent.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/hyper-guardian-agent.md

[6] [7] [28] [29] vault.outpost.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/outposts/vault.outpost.js

[8] [23] [24] dummy-kobalt-schema.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/dummy-kobalt-schema.md

[9] [33] dummy-kobalt-schema.json
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/dummy-kobalt-schema.json

[12] [27] [30] SwarmForm.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/stations/next-visualizer/components/SwarmForm.js

[13] SwarmStatus.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/stations/next-visualizer/components/SwarmStatus.js

[14] [15] SwarmScene.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/stations/next-visualizer/components/SwarmScene.js

[16] [17] predatorPrey.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/eco/predatorPrey.js

[18] [19] alphaEvolveAgent.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/agents/alphaEvolveAgent.js

[26] [31] README.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/README.md