ChatGPT

Spectra CLI – Kobalt Σ Model Overview

Spectra CLI is a comprehensive Unix command-line interface that brings the **Kobalt σ** (Sigma) model to life. It combines generative art and scientific simulations, implementing *swarm* intelligence and *predator-prey* dynamics with sentient, agent-based behavior. This CLI serves as the bridge between the **Arquolab Sandbox** models and a new interactive **Spectra Gallery** toolkit. By exposing core sandbox features across multiple languages (Bash, Python, Node.js, Go, Pascal, Fortran), the Spectra CLI enables a layered, real-time dashboard with form inputs and rich data visualizations. The design is **modular**, **non-entropic** (deterministic and reproducible), and inspired by a dystopian multi-agent ecosystem archetype.

Conceptual Model and Semantics

Kobalt Σ Model: At its core, the CLI is built around the **SigmaAlgeaSwarm** agent (σ) developed by *Kobalt*. This model simulates a boids-like flocking system – agents (or "boids") that follow simple rules of alignment, cohesion, and separation [1] [2]. These rules produce emergent swarm behavior, forming the basis of the "swarm semantics" in our CLI. The Sigma swarm's persistent state (positions, velocities) is stored and continuously updated, reflecting an *agentic*, sentient-like system that evolves over time [3].

Predator–Prey Dynamics: Complementing the swarm is a classic Lotka–Volterra predator-prey model [4], representing ecological interaction semantics. This simulation involves two archetypal agent types – predators and prey – whose populations oscillate as predators hunt prey and prey reproduce [5]. The Arquolab sandbox provides a script (`eco/predatorPrey.js`) that computes these dynamics stepwise [6]. By including predator-prey alongside the swarm, the CLI covers *archetypal typologies* of complex systems: cooperative swarms and competitive survival dynamics. The combination enables exploring scenarios from thriving ecosystems to **dystopian** outcomes (e.g. predator extinction leading to prey overgrowth).

Agentic Archetypes: Several autonomous agents in the sandbox embody different archetypes. For example, **AlphaEvolveAgent** is an evolving neural network model (an "alpha" entity focused on self-improvement), **SigmaAlgeaSwarm** is the collective intelligence archetype (sigma swarm), **QuantumJellyfishAgent** mimics a simple organism with evolving parameters [7] [8], and **HyperGuardianAgent** guards a *hyperbolic universe* simulation via a neural network reacting to sensory input [9] [10]. These agents represent sentient units in our dystopian simulation semantics – each with unique behavior patterns (e.g. evolution, flocking, random drifting, neural response). The CLI unifies control of these agents, allowing users to start/stop them and query status in a consistent way. For instance, the CLI will have commands analogous to the HTTP API endpoints (e.g. `spectra-cli sigma start` to initialize the Sigma swarm just like POST `/api/sigma-swarm/start` in the server [11]).

Archetypal Typology & Semantics: By orchestrating multiple agent types and simulations, Spectra CLI lets users compose scenarios that resemble *archetypal narratives*. One can imagine a **"swarm vs. predator"** scenario or an AI ecosystem where each agent type (Alpha, Sigma, Jellyfish, Guardian) plays a role – cooperation vs. competition, evolution vs. entropy. The semantics of interactions are user-defined via parameters and scenarios, effectively providing a sandbox to explore emergent behavior. The term

"sentient dystopian agentic archetypal typology" in practice means the CLI can simulate a world with autonomous agents (sentient/agentic), facing challenging conditions (dystopian, e.g. resource scarcity or chaotic environments), each agent type representing an archetype (predator, prey, swarm member, guardian, etc.), and users can study the *types* of outcomes or patterns that emerge (typology of simulations).

Features and Components

Multi-Language Support and Extensibility

One of the Spectra CLI's key goals is to be language-agnostic and interoperable across various programming environments: - **Node.js Integration:** The core simulations and agents from Arquolab are written in Node/JavaScript. Spectra CLI uses Node.js under the hood to leverage these modules directly. For example, the CLI can invoke the predator-prey simulation by calling the Node script or function and retrieving results [12], rather than requiring the user to manually run `node eco/predatorPrey.js` and redirect output to CSV as before [13]. - **Python and Data Science Tools:** The CLI can interface with Python for data analysis or advanced scientific libraries. This means after running a simulation, it could optionally call a Python script or use embedded Python (via something like Pyodide or a child process) to perform statistical analysis or generate plots. For instance, the CLI might offer `spectra-cli analyze output.csv --python` to run a Python-based analysis pipeline on simulation data. - **Go Modules:** For performance-critical simulations or concurrency, the CLI could integrate Go. In practice, this could be achieved by calling compiled Go binaries or using cgo bindings. For example, a heavy computation (like a large boid swarm or physics simulation) might be implemented in Go and invoked via `spectra-cli sim --go boids 10000_agents`. - **Pascal/Fortran Support:** To embrace legacy scientific code (common in numerical computing), Spectra CLI is designed to run or wrap Pascal and Fortran programs. Users could plug in a Fortran model (e.g., an old climate simulation) and run it through the CLI, which would handle compiling (if needed) and data piping. This cross-language capability ensures researchers can integrate existing models without rewriting them. *E.g.*, `spectra-cli run fortran model.f90 input.dat` might compile and execute a Fortran program, then feed its output into the CLI's visualization pipeline.

By supporting Bash, Python, Node.js, Go, Pascal, and Fortran, the CLI acts as a **universal orchestrator**. Bash scripts can be used to chain commands or automate sequences (since it's a Unix CLI, it will play nicely with shell piping and scripting). Each language's tool can interface with the CLI via standard I/O or APIs, making the system **polyglot** yet cohesive.

Core Modules and Commands

The Spectra CLI is organized into subcommands corresponding to the main components of the Arquolab sandbox and additional utilities. Key CLI modules include:

- `spectra-cli swarm` – Controls the Sigma swarm simulation. This command can **start**, **stop**, and **monitor** the boid flocking simulation. Under the hood it calls the SigmaAlgeaSwarm agent's methods (`start()`, `stop()`, `status()`) [11] [14]. For example, running `spectra-cli swarm start --numBoids 50 --alignment 1.2 ...` will initialize a swarm of 50 boids with specified parameters, saving state to the database and running the update loop in the background. The CLI will confirm the swarm is running and continuously update its status (e.g., number of boids, simulation step) when queried.
- `spectra-cli sim predator-prey` – Runs the predator-prey population simulation. Users can specify duration or steps (e.g. `--steps 1000`) and initial populations if desired. The CLI

executes the Lotka–Volterra model loop [6], either by invoking the Node script or using an internal function, and outputs the results. Users can choose output formats: table in console, CSV file, or even directly plot (see *Visualization* below). This command encapsulates what the sandbox's `eco/predatorPrey.js` did, but streamlines running and analyzing it.

- `spectra-cli sim chaos-prey` – Executes the more complex **Chaotic Prey-Prey** simulation [15]. This model combines a Lorenz chaotic attractor and a three-body gravitational system influencing two prey populations [16] [17]. The CLI will run the `eco/preyPreyChaos.js` logic and handle output. Because this simulation produces multi-variable output (two prey populations and Lorenz X,Y,Z coordinates per step [17]), the CLI can output a multi-column CSV or even launch a multi-series plot for a quick view of chaotic dynamics. This command demonstrates **layered simulation** support – multiple interacting subsystems (chaos + ecology) in one.
- `spectra-cli agent alpha`, `agent jellyfish`, `agent guardian` – Manage other agents. These commands wrap the respective agents (AlphaEvolveAgent, QuantumJellyfishAgent, HyperGuardianAgent). For example, `spectra-cli agent jellyfish start` will start the JellyfishAgent's internal loop (updating its `topologicalSignature`, `fluxDynamics`, etc. each tick [7] [8]). `spectra-cli agent jellyfish status` would retrieve the latest state (from the MongoDB persistence) and print values like flux or variance. Similarly, `agent guardian start` would initiate the HyperGuardian's neural visualizer (if run headlessly, it might not display graphics but would run the underlying model; in dashboard mode it can display the Three.js scene – see next section).

- `spectra-cli data` – Utilities for data management and projection. Because simulations produce data, this module handles tasks like converting raw output to visualizable format, applying dimensionality reduction, or projecting data onto different spaces. For instance, `spectra-cli data project tensor --dims 12` could load the **Tensor Field** demo (which projects 12-dimensional tensors to 2D) [18]. In CLI context, this might output a static image or open an interactive visualization. The *data* subcommands also include support for seeding random number generators (to ensure runs are deterministic or "non-entropic" when needed), loading external datasets (perhaps feeding real scientific data into simulations), or exporting results to formats for external analysis (Excel, JSON, etc.).

- `spectra-cli dashboard` – Launches the interactive dashboard UI. This command ties together the CLI with a web-based interface (or TUI if in pure console) for a richer experience. Running `spectra-cli dashboard` will start the Next.js **Cosmic Visualizer** server [19] (or a similar web app) and open a browser to interact with simulations in real time. The dashboard provides a **toolbox** of controls and forms to adjust parameters on the fly, along with live visuals (3D and charts). Under the hood, this is achieved by starting the Express/Next server (as in `node index.js` [20]) and possibly using an Electron wrapper for a desktop app or opening the local URL. The CLI ensures that the server has access to all agent endpoints (as defined in `vault.outpost.js`) so that starting a simulation via the form triggers the same functions as the CLI commands.

Dashboard, Toolbox & Forms Interface

Real-time visualization of a predator-prey simulation. The Spectra CLI's dashboard can plot such data live, allowing users to observe oscillations and interact with parameters. In this chart, prey (yellow) and predator (red) populations evolve over time under Lotka–Volterra dynamics.

When the **dashboard mode** is active, users get access to an interactive GUI layered on top of the CLI's capabilities: - **Dynamic Forms:** The interface can generate input forms for simulation parameters using

JSON schemas. For example, a **Kobalt form schema** (derived from design documents) provides dropdowns and fields for configuration [21]. The CLI uses the schema (like `dummy-kobalt-schema.json`) to auto-render forms for selecting topics or setting numeric parameters. This means non-programmers can tweak, say, the number of boids, alignment weight, or predator birth rate through a form instead of command-line flags. The form definitions can be updated by parsing design docs (as done with `generate-kobalt-schema.js` to parse PDFs into JSON schema) [21] [22]. - **Toolbox Controls:** The dashboard provides a toolbox panel (akin to dat.GUI controls or a control panel) for on-the-fly adjustments. Sliders and toggles mapped to simulation parameters enable *layered control* – multiple aspects of the simulation can be tuned simultaneously. For instance, in the tensor field demo, one could adjust dimensions, forces, and oscillation parameters live [23]. In the boids swarm, sliders for alignment/cohesion/separation weights update the swarm behavior in real time [24] [25]. The CLI captures these changes (through its API or direct binding) and applies them without restarting the simulation, illustrating a **non-entropic** control approach (i.e., changes don't inject arbitrary randomness but steer the existing system deterministically). - **Real-Time Data Visualization:** As simulations run, their data streams are visualized. The Spectra CLI dashboard supports multiple synchronized views: - **Time-series charts** (line graphs updating every tick) for things like population vs. time or agent metrics. For example, as the predator-prey simulation runs via CLI, a live plot of prey and predator populations is updated each step, similar to the static graph above but in real time. - **Spatial 2D/3D views** for agent-based models. The Sigma swarm is rendered in 3D via Three.js: each boid is a moving point (or mesh) in space [26] [27]. The HyperGuardian demo renders a dynamic shape in a hyperbolic space. These visuals update continuously (the Next.js app already uses `requestAnimationFrame` to animate [28] [29]). - **Layered Overlays:** The term *layered* refers to stacking multiple visualization layers. Users might overlay the trajectory of one agent type on top of another – e.g. plotting the swarm's spatial distribution alongside a heatmap of predator density. The CLI's web interface can compose layers (e.g., a Three.js layer for geometry, a D3.js or Canvas layer for charts) to provide a composite view. This is facilitated by consistent data sharing via the CLI's backend. For example, the CLI could feed predator-prey data into the same scene as the boids, coloring one boid as a "predator" and others as "prey" in a hypothetical combined simulation. - **Status Monitoring:** The dashboard includes status panels for each running agent or simulation. A **Swarm Status** panel, for instance, shows whether the boid simulation is running and how many boids are active [30]. Similarly, an agent's energy or generation can be displayed and updated every second via polling the CLI's internal state or API [31] [32]. This provides a clear, heads-up display of the simulation's health and progress.

All these GUI components are backed by the CLI's logic and accessible via its API routes. The CLI essentially ensures that for every form action or button click, there's an equivalent CLI command or function being invoked. This dual nature (CLI and GUI) makes the system accessible to both programmers (who prefer terminal commands or scripting) and visual thinkers (who prefer interactive manipulation).

Real-Time Operation and Non-Entropic Design

Real-time feedback is a fundamental aspect of Spectra CLI. Simulations run in continuous loops with a short timestep (often on the order of milliseconds to a second). The Sigma swarm, for example, updates its boids every 1000ms by default [3], and the Quantum Jellyfish and other agents tick once per second updating their state [33]. The CLI is built to handle these updates in real-time by streaming output or using websockets/Server-Sent Events in dashboard mode. This means that if you run a simulation from the CLI without the GUI, you can still watch live text output (e.g., population numbers at each step logged to console). In GUI mode, the same data is pushed to charts or 3D scenes live.

The term **"non entropic"** in the design context emphasizes controlled randomness and reproducibility. While many simulations are stochastic (the swarm initializes boids with random positions/velocities [34],

and the JellyfishAgent adds random perturbations each tick [35]), the CLI provides options to fix random seeds or use deterministic algorithms: - Users can specify a `--seed` for simulations. The CLI will propagate this seed to all random number generators (for example, seeding the PRNG in JavaScript or any Python/Fortran code used). This ensures that runs are repeatable – an essential feature for a scientific data lab approach. - The CLI can utilize **quantum-random stubs** or entropy controls. In the prey-prey chaos model, a crypto PRNG was used for pseudo-quantum randomness [36]. The CLI could allow toggling between true random, crypto RNG, or fixed seeds to see how entropy affects outcomes. - "Non-entropic" also suggests that the system's *information* remains organized rather than dispersing. The CLI is designed to maintain state (via databases or in-memory snapshots) such that users can pause and resume simulations without losing information. For example, SigmaSwarm's state is persisted to MongoDB on every step [37] [38] – if the swarm is stopped and started again, it can continue from where it left off instead of resetting, unless the user explicitly resets it. This persistence fights entropy in the sense of not letting the system devolve into an unknown state over long runs or restarts.

Science Data Lab Projection

A standout feature of Spectra CLI is its ability to function as a **science data lab**, projecting high-dimensional or complex data into interpretable forms. This draws from the sandbox's experimental tools like the tensor field projector and astronomy data integration: - **Multi-Dimensional Data Projection:** The CLI can utilize the **tensor field WebGL shader** to visually project data from many dimensions to 2D [18]. For instance, if a user has a dataset of 12-dimensional state vectors evolving from a simulation, `spectra-cli data project tensor` will open a visualization (using the same logic as `tensorField.html`) where those 12D points are mapped onto a plane with an interactive colorful display. By adjusting parameters (dimensions active, rotation, etc.), the user *projects* the data to glean patterns that aren't obvious in raw numbers. This essentially turns the CLI + dashboard into an interactive *data science lab* for complex systems. - **Integration of External Scientific Data:** Spectra CLI can fetch and incorporate real scientific datasets to augment simulations. The Spectra Gallery's Next.js station already connects to NASA APIs for astronomy imagery [39]. Building on this, the CLI might allow commands like `spectra-cli import nasa-data` or `spectra-cli astro` to bring in planetary data, star charts, or other real-world inputs as initial conditions or comparative datasets. This way, one could project simulation results against real data (e.g., comparing a simulated population model to real ecological data trends) using the same visualization pipeline. - **Analytical Toolbox:** In line with a data lab, the CLI provides analytical tools – for example, computing summary statistics, Fourier transforms of time series, stability analysis of the predator-prey equilibrium, or graphing phase portraits. A user could run `spectra-cli analyze predator-prey --fft` to get a frequency spectrum of the population oscillation, or `--phase-plane prey predator` to plot predator vs prey population (phase plot) for equilibrium analysis. By offering these as built-ins, the CLI saves the user from exporting data to another tool for analysis. - **Collaborative and Reproducible Experimentation:** The lab-like environment fosters reproducibility. Every run can log a configuration and random seed, yielding a "notebook" style record (perhaps output as Markdown or Jupyter notebook). Users can share these or version-control them in the new repository. This echoes scientific lab notebooks, and the CLI could even export a Jupyter Notebook with embedded results for further documentation or publication.

Integration with Arquolab Sandbox and Spectra Gallery

The Spectra CLI is built on the shoulders of the Arquolab Sandbox (the `arquolab-sandbox-models` repository [40]), reusing and repurposing its models in a new context: - **Code Reuse:** The new CLI repository (`spectra-gallery/spectra-cli`) will include the sandbox as a dependency (e.g., as an npm package if published, or a Git submodule). All core logic – simulations, agent classes, helpers – can thus be invoked directly. For example, the CLI can `require('arquolab-sandbox-models/agents/`

`sigmaAlgeaSwarm')` to get the Sigma swarm object and call its methods, or import the `eco/predatorPrey.js` functions. This avoids duplicating code and ensures the CLI benefits from future updates to the sandbox models. - **Spectra Gallery Integration:** As part of the Spectra Gallery ecosystem, the CLI will tie into the gallery's themes and data. It might connect to user accounts or art pieces in the platform (for instance, using the gallery's APIs to fetch generative art assets or to store the outcomes of a simulation as an NFT or artifact). The CLI's visual outputs can be saved and uploaded to the gallery – bridging the gap between running a simulation and curating its results as art/science exhibits. In practice, a command like `spectra-cli export --to-gallery "MySwarmExperiment"` could package the data or animation and send it to the Spectra Gallery web app. - **New Repository Structure:** The new **Spectra CLI** repository will be organized for both CLI distribution and development. It will likely be an npm-based project (for easy installation via `npm install -g spectra-cli`). It may be written in TypeScript or JavaScript for consistency with the sandbox. The repository will include: - A `bin/` script (e.g., `spectra-cli.js`) that initializes the CLI (using a library like Commander.js or Oclif to parse subcommands). - Modules corresponding to each feature area (swarm.js, sim.js, agent.js, data.js, dashboard.js). - A `package.json` referencing `spectra-gallery/arquolab-sandbox-models` (if that is published, or pointing to a Git commit). - Documentation (Markdown files) similar to this one, explaining usage of each command. - Possibly example configuration files or scenario scripts (users can define a sequence of CLI commands in a JSON/YAML to automate experiments). - **Collaboration with Arquolab Sandbox:** Development of the CLI might inform improvements in the sandbox. If the CLI requires certain hooks or more modular functions (for example, a way to step simulations frame-by-frame rather than only running full loops), those could be added to the sandbox models. The two repos will likely remain closely linked. The CLI can be seen as *exposing* the sandbox's capabilities to end users in a friendly format, whereas the sandbox itself is a developer-oriented collection of models.

Example Usage Scenarios

To illustrate how researchers, artists, or developers might use the Spectra CLI, here are a few example scenarios and commands:

- **Ecosystem Simulation and Visualization:** A user wants to explore predator-prey dynamics and swarm behavior together. They run:

  ```
  spectra-cli swarm start --numBoids 30 --alignment 1 --cohesion 1 --
  separation 1.5 &
  spectra-cli sim predator-prey --steps 500 --output predator_prey.csv
  spectra-cli dashboard
  ```

  This starts a boid swarm in the background, runs a 500-step predator-prey simulation (saving data to CSV), and then launches the dashboard. In the dashboard, the user loads the `predator_prey.csv` data (or sees it already plotted as a time series) and also sees the boids moving in real-time. They could then interactively adjust the swarm parameters via sliders – for instance, increasing the separation weight to see the flock disperse more [41] – and observe how it doesn't directly affect the predator-prey chart but might in a future combined model (conceptually linking the two systems).
- **Agent Monitoring and Control:** An AI researcher uses the CLI to monitor the *AlphaEvolveAgent* and *HyperGuardianAgent*. They run:

```
spectra-cli agent alpha start
spectra-cli agent guardian start
watch -n 1 spectra-cli agent alpha status
```

This starts the Alpha evolving agent and the Guardian visual agent. The `watch` command (standard Unix) then polls the CLI every 1 second for the alpha agent's status. Each poll prints something like:
`Generation: 42, RotationRate: 0.5, NeuralWeights: [...]` (assuming status output). Meanwhile, the guardian agent may be affecting a visual (if dashboard is open) or simply running a neural net. After some time, the researcher stops them with `spectra-cli agent alpha stop && spectra-cli agent guardian stop`. All the while, the CLI ensured that these agents' states were saved in MongoDB, so restarting them resumes from last state [42] [43].

- **Data Analysis Workflow:** A data scientist wants to use the CLI purely as a data tool. They feed in an external dataset to compare with a simulation. For example:

```
spectra-cli sim predator-prey --steps 300 --output sim.csv --seed 123
spectra-cli data import real_ecosystem.csv --name real
spectra-cli data compare sim.csv real --metrics population,frequency
spectra-cli data project sim.csv --pca 2D
```

In this sequence, the CLI runs a predator-prey model with a fixed seed for reproducibility, producing `sim.csv`. Then it imports a real dataset (perhaps historical predator-prey counts from nature) and assigns it a label. The `compare` subcommand could compute differences in metrics (like average population, or frequency of oscillation) between the simulation and real data. Finally, `data project --pca 2D` might perform a principal component analysis to project potentially high-dimensional features of the simulation into 2D for visualization, or simply plot prey vs predator counts as a phase plot. The CLI thus serves as a mini data science pipeline, all reproducible via command-line instructions.

- **Generative Art Creation:** An artist can leverage the CLI for creative coding. Suppose they want to generate a visualization of a "dystopian swarm" for an art piece. They might do:

```
spectra-cli swarm start --numBoids 100 --alignment 0.8 --cohesion 0.2 --
separation 2.0 --seed 999
spectra-cli dashboard --headless --record swarm.mp4 --duration 60
```

Here, the user starts a swarm of 100 boids with certain weights that cause a very dispersive behavior (high separation, low cohesion, giving a chaotic look). They then launch the dashboard in headless mode with recording enabled – the CLI runs the visualization for 60 seconds and records the 3D canvas to a video file `swarm.mp4`. The artist can later overlay this with dystopian soundscapes or use it in multimedia projects. Because the CLI allowed a seed, if they need to regenerate the exact same visual (same initial conditions), they can. Alternatively, they can try different seeds to find a pattern they like, knowing each seed yields a deterministic but distinct swarm motion.

Conclusion

The **Spectra CLI** is a unifying interface that transforms the rich **Arquolab Sandbox** into an accessible tool for exploration and creation. By implementing the Kobalt σ model and a suite of archetypal simulations, it enables users to dive into swarm intelligence, predator-prey ecosystems, chaotic dynamics, and evolving agents — all through a single cohesive CLI. The integration of a real-time dashboard with forms and visualization means users can seamlessly transition from command-line experiments to interactive "science lab" sessions. Moreover, the CLI's multi-language interoperability ensures that whether one's asset is a Node.js agent, a Python analysis script, or a legacy Fortran model, it can plug into the Spectra ecosystem.

In essence, Spectra CLI embodies a *sentient laboratory* for both art and science: it treats code and data as living organisms (agents and populations) that one can manipulate and observe. The dystopian undertones (unpredictable emergent behavior, the mix of biological and artificial "life") serve as a creative backdrop, while the rigorous support for data projection and reproducibility grounds it in real scientific methodology. By open-sourcing this tool in a new repository and aligning it with Spectra Gallery, the project invites collaboration from developers, researchers, and artists alike – pushing the boundaries of creative coding, generative art, and complex systems simulation in an accessible, powerful way.

Sources: The design and implementation details above are drawn from the Arquolab sandbox code and documentation. Key references include the Sigma swarm agent code [1] [2], agent architecture docs [44], predator-prey and chaos simulation scripts [4] [15], and the Spectra whitepaper's outlined demos [18], among others. All these components come together to inform the architecture of the Spectra CLI.

---

[1] [2] [3] [34] [37] [38] [41] sigmaAlgeaSwarm.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/agents/sigmaAlgeaSwarm.js

[4] [5] [6] predatorPrey.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/eco/predatorPrey.js

[7] [8] [33] [35] quantum-jellyfish-agent.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/quantum-jellyfish-agent.md

[9] [10] hyper-guardian-agent.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/hyper-guardian-agent.md

[11] [14] vault.outpost.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/outposts/vault.outpost.js

[12] [13] [15] [18] [23] whitepaper.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/whitepaper.md

[16] [17] [36] preyPreyChaos.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/eco/preyPreyChaos.js

19 39 README.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/stations/next-visualizer/README.md

20 40 FAQ.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/FAQ.md

21 22 dummy-kobalt-schema.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/dummy-kobalt-schema.md

24 25 SwarmForm.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/stations/next-visualizer/components/SwarmForm.js

26 27 28 29 SwarmScene.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/stations/next-visualizer/components/SwarmScene.js

30 31 32 SwarmStatus.js
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/stations/next-visualizer/components/SwarmStatus.js

42 43 44 agent-architecture.md
https://github.com/spectra-gallery/arquolab-sandbox-models/blob/6ef3a9408d80574cfed983ea40b8c17a597b9acc/docs/agent-architecture.md