ChatGPT

Hyperon/MeTTa Core & Architecture

OpenCog Hyperon is a new AGI framework built around the MeTTa (Meta Type Talk) language [1] [2]. MeTTa is a highly abstract, multi-paradigm language where *every program is a subgraph of a dynamic "Atomspace" metagraph*, enabling pattern-matching queries, rewriting, and self-modifying code at runtime [3]. It blends functional, logical, probabilistic, and even neural-symbolic paradigms [4] [5]. For example, MeTTa supports *gradual dependent types* for built-in mathematical reasoning, and tightly integrates uncertain inference (Probabilistic Logic Networks, fuzzy logic, etc.) with symbolic data [6] [7]. Programs can read and modify their own code, creating native support for self-optimization and reflection [3].

The Hyperon implementation is primarily in Rust. The core library **libhyperon** (built with Cargo) is divided into several crates [2]: **hyperon-common** (general utilities and data structures), **hyperon-atom** (the core Atom and Atoms-space APIs for built-in modules), and **lib** (the MeTTa Atomspace storage and interpreter). A C API (**libhyperonc** in the `/c` folder) exposes the core to other languages, and Python bindings are provided via a pybind11 layer (**libhyperonpy** and the `hyperon` Python package) [8]. In practice one can embed Hyperon in Rust, Python, or C/C++ code. (For IDE support, Hyperon uses Rust analyzer, Clangd, and Python LSP servers [9].)

High-Performance Kernel (MORK)

A major Hyperon subproject is **MORK** ("MeTTa Optimal Reduction Kernel"), a Rust-based hypergraph engine and virtual machine designed to accelerate MeTTa execution [10]. MORK implements a high-performance graph database with a "zipper-based" multi-threaded VM to dramatically speed up graph transformations, unification, and rule application [10]. Its goal is to eliminate bottlenecks in basic Atomspace operations so that MeTTa reasoning can scale (potentially thousands to millions of times faster) on large knowledge spaces [11]. In other words, MORK re-architects Hyperon's storage and execution layer for extreme throughput (analogy: exposing full hardware parallelism for symbolic AI, as GPUs did for deep learning) [11]. The MORK repository contains a `kernel/` subproject that builds a command-line VM (`cargo build --release` in `/kernel` using Rust nightly) [12]. (MORK is still experimental; the main README [36] describes it as "pre-alpha" and directs power users to its `server` branch or documentation.)

Inference & Learning Modules

Hyperon's design encourages modular inference algorithms. Several active repos implement classic OpenCog algorithms on Hyperon:

- **Pattern Miner** – The **hyperon-miner** project is a *port of OpenCog's pattern miner* to Hyperon/MeTTa [13]. Pattern mining discovers frequently occurring subgraph patterns in the Atomspace, which can seed new knowledge. The hyperon-miner code (mostly Prolog, with some Python and shell) implements this mining engine on Hyperon [13].

- **Chaining** – The **chaining** repo provides a MeTTa module with *backward and forward chaining inference*. Its README says: "various flavors of backward and forward chaining, as well as converters between these flavors" are implemented [14]. In practice, these are graph-rewriting rules (e.g. modus ponens, forward inference) written in MeTTa/Prolog that traverse implication links in the Atomspace. The chaining code is a collection of MeTTa programs (shell/Prolog) for doing both proof-as-program execution and proof-as-program-trace inferences [14].

- **Probabilistic Logic Networks (PLN)** – There are two PLN-related projects:

- **PLN (Shell)** – A "Modern PLN implementation for Hyperon" (in a shell-based repo named **PLN**, not to be confused with Classic OpenCog) [15]. This contains scheme/shell scripts realizing uncertain inference rules (Bayes/Implication, P-completeness, etc.) adapted to Hyperon.

- **hyperon-pln (Idris)** – An Idris-based implementation of PLN ("Hyperon port of PLN") [16]. This appears to be a rewrite of PLN in Idris, possibly for formal verification or new performance. (The original **pln-experimental** repo is a Dockerfile and docs, now superseded by the live **PLN** repo.)

- **MOSES (AsMODS)** – The **hyperon-moses** repo is a Hyperon port of *ASMoses* (the OpenCog meta-optimizing evolutionary learner) [17]. Its description simply states "Hyperon port of asmoses." This implements the genetic programming / machine learning system on Hyperon's Atomspace. (Details must be in code; the repo indicates AGPLv3 but we have only the summary line [17].)

- **Other** – The Hyperon ecosystem expects other cognitive modules (ECAN/attention, etc.), but no specific repo was found beyond above. The **metta-catalog** repo exists as an *index of available MeTTa modules* (no code) for tracking libraries.

Language Integrations and Tools

Hyperon/MeTTa is accessed and extended in multiple ways:

- **MeTTaLog (metta-wam)** – The **metta-wam** project is a Prolog-based interpreter/compiler for MeTTa targeting the Warren Abstract Machine (WAM) [18] [19]. Its README calls it "An implementation of MeTTa designed to run on the WAM." This code (SWI-Prolog with Python glue) allows MeTTa code to be executed via Prolog's inference engine, using the Janus or pyswip interfaces. It includes a MeTTa grammar, compiler to Prolog, Jupyter kernel (`mettalog-vspace`, `mettalog-jupyter-kernel` are installed in setup [20]) and testing suite. In effect, metta-wam is an alternate (non-Rust) runtime for MeTTa/Hyperon, useful for experimentation and for running legacy "MeTTaLog" programs.

- **Jupyter Kernel (PeTTa)** – The **jupyter-petta-kernel** repo provides an *IPython kernel for MeTTa*, called "PeTTa." Its README states: "A Jupyter kernel for executing MeTTa code using PeTTa." In practice, PeTTa is the name of a Python implementation of the MeTTa VM. The kernel wraps this: it requires installing PeTTa separately, then installing the kernel spec [21]. Once registered, one can create Jupyter notebooks with a "PeTTa (MeTTa)" kernel, write MeTTa code in cells, and execute it interactively [22]. (This tool is helpful for development and tutorials.)

- **Metta-Morph** – The **metta-morph** project is a *macro-based MeTTa→Scheme translator*. Its README says "Metta-morph (from Metamorphosis): Macro-based MeTTa to (Chicken) Scheme translator." Essentially it compiles MeTTa code into equivalent Scheme code (for the Chicken Scheme dialect), using syntax macros. This can serve as an alternate backend or a demonstration of MeTTa's extensibility [23].

- **protobuf-metta** – The **protobuf-metta** repo provides a generator from Google Protocol Buffer definitions to MeTTa code. Its description says "Convert protobuf specification to MeTTa" [24]. This tool likely auto-generates MeTTa type and message definitions from `.proto` files, easing integration with external data formats.

- **Other Utilities** – A few smaller repos support the ecosystem. For example, **metta-catalog** lists available MeTTa modules (currently a stub index). The **metta-examples** repo contains annotated example programs and discussions (a tutorial resource) [25].

Example Applications (Minecraft Demos)

TrueAGI has released experimental demo projects illustrating Hyperon/MeTTa in action:

- **Vereya** – A Python API for Minecraft agents. The **Vereya** repository "is a Python API for agents in Minecraft" [26]. It likely wraps Hyperon-based decision-making for Minecraft bots (though the repo shows it's a Java/Gradle project, the description highlights Python).

- **Minecraft-Demo** – The **minecraft-demo** repo contains a high-level API to interact with a Minecraft client library (AGPLv3) [27]. The README tagline is "High-level api for interaction with the client library," and the code likely uses Hyperon logic to control a Minecraft agent.

- **Minecraft-Experiments** – The **minecraft-experiments** repository (Python) contains sample programs and notebooks for Minecraft scenarios. It appears to include scripts for agent behaviors, block analysis, and observations (folders like `behavior`, `block_analysis`, `examples/observations` are in the repo) [28]. Together, these repos show how Hyperon and MeTTa can be applied to game-world embodied AI, but primarily serve as examples/demos.

Summary of Active Repositories

In summary, the TrueAGI/Hyperon ecosystem includes (among others):

- **hyperon-experimental** – Main Hyperon implementation (Rust core library, Atomspace, interpreter).
- **MORK** – High-performance hypergraph engine for Hyperon (Rust) [10].
- **hyperon-miner** – Pattern mining (Prolog/MeTTa) [13].
- **chaining** – Forward/backward chaining inference (MeTTa modules) [14].
- **PLN** – Shell-based Probabilistic Logic Networks for Hyperon [15].
- **hyperon-pln** – Idris port of PLN (experimental).
- **hyperon-moses** – Hyperon port of ASMoses [17].
- **metta-wam** – Prolog/WAM MeTTa interpreter (MeTTaLog) [19].

- **jupyter-petta-kernel** – Jupyter notebook kernel for MeTTa [21].
- **metta-morph** – MeTTa→Scheme macro translator [23].
- **protobuf-metta** – Protobuf→MeTTa code generator [24].
- **metta-examples** – Tutorials and example programs [25].
- **Vereya**, **minecraft-demo**, **minecraft-experiments** – Minecraft AI demos [26] [27].
- **metta-catalog** – (Stub for indexing MeTTa modules).

Each repo's README provides details on its implemented features (as cited above). For example, the chaining repo's title explicitly lists "backward and forward chaining" algorithms [14], and the miner's README states it's a port of the OpenCog Pattern Miner [13]. In aggregate, this ecosystem covers the core MeTTa execution engine plus a suite of AI algorithms (uncertain inference, logic chaining, evolutionary learning, pattern mining, etc.) implemented on top.

Sources: TrueAGI GitHub repos and documentation [2] [3] [10] [13] [14] [19] [21] [27] [17] [24] [15] [26] (see links). These show the available code, descriptions, and licenses for each component.

---

[1] [3] [4] [5] [6] [7] MeTTa Programming Language - ASI | Artificial Superintelligence Alliance
https://superintelligence.io/portfolio/metta-programming-language/

[2] [8] [9] GitHub - trueagi-io/hyperon-experimental: MeTTa programming language implementation
https://github.com/trueagi-io/hyperon-experimental

[10] [11] [12] GitHub - trueagi-io/MORK: MeTTa Optimal Reduction Kernel
https://github.com/trueagi-io/MORK

[13] GitHub - trueagi-io/hyperon-miner
https://github.com/trueagi-io/hyperon-miner

[14] GitHub - trueagi-io/chaining: MeTTa module containing various flavors of backward and forward chaining, as well as converters between these flavors.
https://github.com/trueagi-io/chaining

[15] trueagi-io · GitHub
https://github.com/trueagi-io

[16] tanksha (Hedra S Yusuf) · GitHub
https://github.com/tanksha

[17] [24] trueagi-io repositories · GitHub
https://github.com/orgs/trueagi-io/repositories

[18] [19] [20] GitHub - trueagi-io/metta-wam: A Hyperon MeTTa Interpreter/Transpilier that targets the Warren Abstract Machine
https://github.com/trueagi-io/metta-wam

[21] [22] GitHub - trueagi-io/jupyter-petta-kernel
https://github.com/trueagi-io/jupyter-petta-kernel

[23] GitHub - trueagi-io/metta-morph: Metta-morph (from Metamorphosis): Macro-based MeTTa to (Chicken) Scheme translator.
https://github.com/trueagi-io/metta-morph

25 GitHub - trueagi-io/metta-examples: Discussion of MeTTa programming with examples
https://github.com/trueagi-io/metta-examples

26 GitHub - trueagi-io/Vereya: Python api for agents in minecraft
https://github.com/trueagi-io/Vereya

27 GitHub - trueagi-io/minecraft-demo
https://github.com/trueagi-io/minecraft-demo

28 GitHub - trueagi-io/minecraft-experiments
https://github.com/trueagi-io/minecraft-experiments