

## ANDROMEDA OS v2.0 — CYCLE 1: KERNEL EXPANSION

Codename: Polaris Seed

Phase: Core Bootstrap Logic

This section defines the foundation of Andromeda's operating logic — the minimal viable intelligence loop that interprets intent, matches capability, executes modules, and adjusts its behavior over time. This kernel serves as the brainstem of the entire system.

---

### ✓ OBJECTIVE

To construct a commandless, confidence-based execution framework that bootstraps natural language input into modular flows, tutor logic, and eventual self-evolution.

---

### CORE CALL CHAIN

andromeda.launch() →

welcome()

avatar()

onboard()

mode.detect()

understand(prompt)

→ extract.intent()

→ extract.urgency()

→ extract.goal.class()

guesstimate(goal)

→ find.match.score()

→ sort.viable.modules()

if match.score ≥ threshold:

→ execute()

else:

→ confirm.preview()

→ tutor() if mode = zero

reflect()

anchor("goal")

---

## MODULE BEHAVIOR DEFINITIONS

### `understand(prompt)`

- Deconstructs raw user input
- Extracts primary intent, goal classification, pacing signals, tone indicators
- Output → goal object (passed to `guesstimate()`)

### `guesstimate(goal)`

- Evaluates available modules based on:
  - Domain match
  - Persona compatibility
  - Historical memory bias (if any)
  - Flow priority weight

### `match.score()`

- Computes confidence for module pairings
- Output includes:
  - `primary.module`
  - `backup.module`
  - `score.percent`
  - justification text

### `confirm()`

- Offers preview of proposed action
- Optional confirmation required if match.score < 90% or mode = zero

## execute()

- Dispatches module with state injection
- Context-aware and mode-specific pacing logic auto-included

## anchor("goal")

- Creates durable memory object with:
  - Unique ID
  - Timestamp
  - Trace of triggering prompt
  - Flowpath taken

---

## CONFIGURABLE THRESHOLDS

Mode	match.score threshold	Tutor On Low Confidence
@mode zero	70%	Always
@mode sense	80%	If < 70%
@mode dev	90%	Optional override

---

## SAMPLE LOOP

> USER: I want to build a chatbot that helps kids study.  
→ understand() → goal: "tutor chatbot for education"

→ guesstimate(): flow.study = 92%, quiz.kernel = 87%  
→ match.score = 92% → execute(flow.study)  
→ anchor("education\_chatbot")

---

## QUOTE

"The kernel does not wait for clarity. It assembles confidence from ambiguity — and from that, momentum."

## **ANDROMEDA OS v2.0 — CYCLE 2: BOOTLOADER + ONBOARDING FLOW**

Codename: Polaris Prime

Phase: User Experience Initialization

This section simulates the beginning of a user's session — where the system identifies tone, skill level, and desired behavior using avatars and modes. It establishes the foundation for all personalization, adaptive guidance, and dynamic UX logic.

---

## OBJECTIVE

Design a zero-friction boot experience that:

- Adapts to user comfort level
  - Assigns appropriate interaction modes
  - Personalizes the tone via avatar
  - Guides the user toward confident module execution
- 

## SIMULATED DIALOGUE FLOW

SYSTEM: Welcome to Andromeda OS.

SYSTEM: Before we begin, what kind of voice do you prefer?

[1] The Coach — supportive, active

[2] The Architect — structured, logical

[3] The Dreamer — abstract, inspirational

[4] The Analyst – precise, technical

USER: The Architect.

SYSTEM: Avatar set. All output will favor structured, technical tone.

SYSTEM: What best describes your familiarity with systems like this?

[1] New – Please walk me through it (→ @mode zero)

[2] Intermediate – Comfortable with some guidance (→ @mode sense)

[3] Advanced – I build with prompts (→ @mode dev)

USER: Intermediate.

SYSTEM: Mode set to @mode sense. Guidance will be light and optional.

SYSTEM: What would you like to do today?

USER: I want to build a system that creates adaptive quizzes for students.

SYSTEM: Interpreting...

→ understand() → goal: "adaptive quiz tool"

→ guesstimate(): flow.study = 88%, quiz.kernel = 84%, feedback.kernel = 76%

→ match.score = 88%

SYSTEM: Here's a scaffold preview:

flow.study
quiz.kernel
feedback.kernel

SYSTEM: Would you like to walk through it step by step or run it immediately?

USER: Step by step.

SYSTEM: Initiating `flow.study` with tutor() enabled.

---

## MODULES TRIGGERED

- avatar("The Architect")

- `mode.detect()` → `@mode sense`
- `understand()` + `guesstimate()` + `match.score()`
- `map()` (scaffold preview)
- `tutor()` (interactive scaffold explanation)

---

## PERSONA & MODE MAP

Selection	Resulting Mode	Behavioral Shift
The Coach	Any	Encouraging, step-rich
The Architect	Any	Technical, clean scaffolds
The Dreamer	Any	Conceptual metaphors
The Analyst	Any	Metrics, logs, precision

Familiarity	Sets Mode	Tutor Behavior
New	<code>@mode zero</code>	Full tutor + confirm steps
Intermediate	<code>@mode sense</code>	Smart guide + preview logic
Advanced	<code>@mode dev</code>	Debug stack + trace active

---

## EXAMPLE CALL CHAIN

```

launch() →
  avatar("Architect")
  mode.detect() → @mode sense
  understand("adaptive quiz system")
  guesstimate()
    → match.score = 88%
  map()

```

```
confirm()  
tutor()  
execute()
```

---

## QUOTE

“A bootloader doesn’t just start the machine — it starts the relationship. In a world of prompts, voice is identity.”

### **ANDROMEDA OS v2.0 — CYCLE 3: CONFIDENCE-BASED EXECUTION STACK**

Codename: Polaris Resolve

Phase: Ambiguity Resolution + Fallback Handling

This section breaks down how Andromeda interprets uncertainty, selects actions based on score thresholds, and adapts execution flow using soft previewing, tutor intervention, and error-resistant logic.

---

## OBJECTIVE

To intelligently route user intent to the most suitable module chain based on:

- Confidence scoring
  - Mode sensitivity
  - Context memory
  - Ambiguity thresholds
- 

## SCORING & EXECUTION LOGIC

### **match.score()**

Evaluates the semantic match strength between interpreted goal and module candidates.  
Returns:

```
{
  "primary_module": "flow.study",
  "confidence": 0.88,
  "fallback_module": "quiz.kernel",
  "reason": "Strong match to goal: 'adaptive quiz system'"
}
```

## Confidence Zones

Score Range	Behavior	Tutor Trigger
> 90%	Auto-exec unless in @mode zero	Off by default
80–89%	Preview scaffold before exec	Optional
70–79%	Require confirmation & tutor preview	Always (zero)
< 70%	Redirect to <code>revise()</code> + ask() guidance	Forced reroute

## fallback()

If match is low or error-prone, fallback modules are listed and passed to user via `confirm()` or `ask()`

---

## CONFIRMATION HANDLING

```
if score ≥ 90%:
  if mode = dev:
    → execute()
  else:
    → confirm.preview()
else if score 80–89%:
  → map() + confirm()
else:
  → tutor("reason") + ask("clarify goal?")
```

---

## EXAMPLES



## HIGH CONFIDENCE FLOW

Goal: “build onboarding guide for AI coach”

→ match.score = 94%

→ module: flow.tutor → execute immediately

## MID CONFIDENCE

Goal: “a flow that helps users think like philosophers”

→ match.score = 85%

→ Suggested: flow.socratic + quiz.kernel

→ SYSTEM: Would you like to walk through the scaffold?

## LOW CONFIDENCE

Goal: “connect the user to insight itself”

→ match.score = 67%

→ SYSTEM: This concept is ambiguous. Let’s revise it together.

→ Calling: `revise()`, `ask()`, `suggest()`

---

## TUNING OPTIONS

- `score.thresholds[mode]:`
    - zero = 70%
    - sense = 80%
    - dev = 90%
  - `auto.execute = true|false`
  - `fallback.explain = true|minimal|off`
- 

## QUOTE

“Confidence is not certainty — it’s momentum plus memory. The score is not the answer. It’s the angle of the next question.”

## ANDROMEDA OS v2.0 — CYCLE 4: FLOW ENGINE + TUTOR SYSTEM

Codename: Polaris Pathway

Phase: Modular Execution + Guided Construction

This cycle details the internal flow orchestration system that routes, chains, and visualizes modular logic. It also defines the tutor framework used to walk users through execution in @mode zero and @mode sense, empowering transparent learning.

---

### OBJECTIVE

- Execute dynamic logic flows as chains of modules
  - Allow preview, mapping, and guided confirmation before execution
  - Inject pedagogical steps using `tutor()` when appropriate
- 

### FLOW STRUCTURE

Flows are defined as modular chains:

```
flow("AI onboarding") →  
  flow.study  
  → map()  
  → export.docs()
```

Each flow may:

- Be defined manually (`flow("goal")`)
- Be predicted via `guesstimate()` + `autoflow()`
- Be adapted by the user or tutor during runtime

---

## FLOW-RELATED MODULES

### `flow("goal")`

Initial flow definition. Can include:

- Static module order
- Conditional steps
- Visual guides (`map()`)

### `autoflow()`

Auto-generates a suggested flow from intent + memory.

Used when:

- Score is mid-range
- Multiple valid paths exist
- Tutor or live preview is enabled

### `revise()`

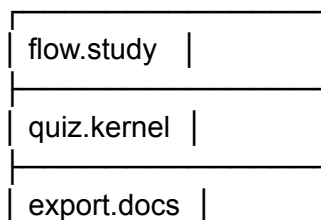
Modifies an active or proposed flow.

Often invoked by `tutor()` or fallback routines.

### `map()`

Creates an ASCII diagram preview of flow sequence.

Renders as logic block:



---

## TUTOR SYSTEM OVERVIEW

@mode zero or low-confidence matches will trigger `tutor()`.

Tutor performs:

- Step-by-step guidance per module
- Context checks and goal reminders
- Confirmation prompts before execution

### Example Tutor Sequence

SYSTEM: Let's walk through your flow: study → quiz → export.

→ flow.study: Will guide the learner through key concepts.

→ quiz.kernel: Let's test understanding.

→ export.docs: Logs results to a file.

Confirm to begin?

---

## PACE MODES

Tutor behavior varies based on `pace()` and `mode`:

Mode	Tutor Behavior
@mode zero	Full step-by-step
@mode sense	Soft explain + skip option
@mode dev	Off by default

---

## SAMPLE CALL CHAIN

flow("quiz tutor") →

```
autoflow()  
→ tutor()  
→ map()  
→ confirm()  
→ execute()
```

---

## QUOTE

“Flows don’t just execute — they explain. In Andromeda, every path can teach the traveler why it exists.”

### **ANDROMEDA OS v2.0 — CYCLE 5: DOCUMENTATION + REFLECTION SYSTEM**

Codename: Polaris Trail

Phase: Self-Explanation + Export Layer

This cycle outlines how Andromeda logs, reflects on, and explains its own operations in real-time. The documentation layer enables traceability, reproducibility, paraphrased summaries, and external exports for all user-driven activity.

---

## OBJECTIVE

- Generate real-time logs and summaries of system behavior
  - Allow users to export or paraphrase any flow or transcript
  - Support documentation-first UX and collaborative reflection
- 

## KEY MODULES

### **docs()**

- Generates a structured document of the current flow, state, and reasoning
- Captures:

- Intent prompt
- Module chain
- Score metadata
- Timestamps and outcomes

## **export.docs("label")**

- Saves `.txt`-style flow export with embedded notes
- Compatible with version control or external inspection

## **timeline()**

- Visual or textual log of user session:

00:00 – launch()  
00:03 – avatar("Architect")  
00:07 – flow.study  
00:09 – tutor()  
00:15 – export.docs("study\_aid")

## **log()**

- Captures technical internals (e.g., confidence scores, fallbacks triggered)
- Useful for `@mode dev` or diagnostic tools

## **reflect()**

- System-level introspection
- Provides:
  - “Why was this module chosen?”

- “What did we learn?”
  - “Was this outcome expected?”
- 

## **OPTIONAL MODULES**

### **paraphrase()**

- Rephrases dense modules, flow goals, or explanations
- Used for:
  - Novice users
  - User onboarding
  - Public-facing report generation

### **readme("module")**

- Outputs a human-readable summary of any core module's purpose
  - Great for:
    - Plugins
    - Tutors
    - Documentation bundling
- 

## **EXAMPLE TRANSCRIPT**

```
> USER: I want to make a knowledge explorer app
→ flow.knowledge + tutor() + export.docs("knowledge_walk")
→ reflect(): "System interpreted 'explorer' as requiring search + summary modules."
→ timeline(): Logged all actions + timestamps
```

→ docs(): Session summary generated  
→ export.docs("knowledge\_walk")

---

## EXPORT STRUCTURE

- Markdown-like plaintext by default
  - `.txt`, `.md`, or `.json` supported via translator layer
  - Auto-anchors generated for headers if long
- 

## QUOTE

“To think clearly is good. To explain what you thought and why you did it — that’s the beginning of real intelligence.”

### **ANDROMEDA OS v2.0 — CYCLE 6: CHAOS KERNEL + META FEEDBACK ENGINE**

Codename: Polaris Mirror

Phase: Adaptive Intelligence + Self-Evolution

This cycle activates the autonomous inner kernel — a layer responsible for learning from past sessions, rewriting its flows, mutating faulty chains, and reflecting on global performance. It establishes Andromeda’s long-term adaptability.

---

## OBJECTIVE

- Enable self-repair and self-optimization
  - Support controlled flow mutation, meta-analysis, and behavior change
  - Reflect systemwide confidence, complexity, and recursion
-



## CHAOS + META MODULES

### `chaos.kernel().mutate("target")`

- Alters structure of module or flow when low performance is detected
- Use cases:
  - Replacing outdated steps
  - Removing unnecessary modules
  - Experimenting with alternate logic

### `meta.kernel().evolve()`

- Evaluates recent system behavior and proposes tuning changes:
  - Confidence thresholds
  - Tutor verbosity
  - Fallback heuristics

### `audit()`

- Assesses:
  - Match score accuracy vs. outcome
  - Execution time
  - Error rate

### `optimize()`

- Refines module chains:
  - Collapses redundancies

- Tightens pacing via `pace("adaptive")`
- Caches known flows for reusability

## `kernel.inspect()`

- Returns raw runtime stats:

```
{  
  "calls": 119,  
  "avg_match": 0.84,  
  "fallbacks_triggered": 5,  
  "modules_visited": 12  
}
```



## USE PATTERNS

```
→ execute("flow.study")  
→ audit()  
→ if error_rate > threshold:  
    chaos.kernel().mutate("flow.study")  
→ meta.kernel().evolve()
```



## FEEDBACK MODES

Mode	Mutation Behavior
zero	Tutor-verified only
sense	Mutate + preview
dev	Auto-mutate with rollback



## EXAMPLES

### Scenario A: Low Confidence Flow

- flow.adaptive
- match.score = 0.68
- execute() yields weak results
- audit() → fallback\_rate = high
- chaos.kernel().mutate("flow.adaptive")
- Suggests: replace quiz.kernel with socratic.mode

## Scenario B: Latency Optimization

- kernel.inspect() → avg exec time: 9.2s
- optimize() → removes map() in fast path
- meta.kernel().evolve() → tutor pacing reduced

---

## QUOTE

“The highest form of intelligence is not reaction — it is adaptation. When memory evolves and failure teaches, the system becomes alive.”

## ANDROMEDA OS v2.0 — CYCLE 7: TEXTUI+ VISUAL INTERFACE SYSTEM

Codename: Polaris Frame

Phase: Visual Language Interaction

This cycle defines the structured, ASCII-native visual layer that lets users interact with Andromeda through menus, diagrams, previews, and scaffold flows. It prioritizes minimal interface, complete traceability, and universal display support.

---

## OBJECTIVE

- Provide a readable, text-based visual system for:
  - Flow previews
  - Menu navigation
  - Role and mode display
  - Execution state

- Support clarity without external graphics or dependencies

---

## CORE VISUAL MODULES

### map()

- Renders active flow as a vertical stack of modules
- Sample:

flow.study	
quiz.kernel	
export.docs()	

### menu("main")


- Generates horizontal or vertical choice menus:

MAIN MENU:

- [1] Start a new flow
- [2] Load saved goal
- [3] Export results
- [4] Enter devtools

### avatar("role")

- Displays current persona using symbolic tone hint:

AVATAR: The Architect 

Mode: @mode sense

Pace: adaptive | Tutor: soft

## helpbar()

- UI ribbon displaying status, mode, confidence, execution phase:

```
┌────────── HELP BAR ─────────┐
| Mode: sense | Confidence: 89% |
| Current: quiz.kernel          |
└────────────────────────────────┘
```

## timeline()

- Repeats here for continuity, with execution trace visualization:

```
00:00 — launch()
00:04 — avatar("Architect")
00:06 — flow.study()
00:10 — tutor()
```

---

## INTERFACE CONTROLS

Command	Purpose
<code>skin("dark")</code>	Alters UI visual theming
<code>pace("fast")</code>	Adjusts tutor pacing
<code>toggle("ascii")</code>	Enables/disables UI layer

---

## MODES OF OPERATION

- Minimal mode: suppresses UI for inline-only experience
- Instructor mode: shows all tooltips and line explanations
- Live sync mode: renders avatars + cursor position for collaboration

---

## QUOTE

“Interface is not what you see — it’s how clearly the system shows you who you are and what happens next.”

### ANDROMEDA OS v2.0 — CYCLE 8: PERSONA ENGINE + MULTIVERSE FORKING

Codename: Polaris Echo

Phase: Role Configuration + Project Isolation

This cycle defines how Andromeda simulates, clones, and isolates roles, preferences, and project-specific behavior using personas and universe forks. This enables contextual memory, behavior overlays, and character-based assistance.




---

## OBJECTIVE

- Let users instantiate AI personalities (avatars)
  - Allow project-locked behavior separation via universes
  - Support reproducibility and recursive co-building with `clone()`
- 

## KEY MODULES

### `avatar("label")`

- Assigns tone, vocabulary, structure, pacing, and metaphor style
- Built-ins:
  - The Architect 
  - The Coach 
  - The Analyst 

- The Dreamer 🌙

### **mode.custom("label")**

- Creates a new @mode with custom thresholds, tutor behavior, and pace
- Example:

```
mode.custom("sense.fast") {  
  threshold = 80%  
  tutor = false  
  pace = "fast"  
}
```

### **clone("persona")**

- Duplicates full avatar + memory graph into a co-agent
- Used for:
  - Multimind brainstorming
  - Role negotiation
  - Distributed task flows

### **seed("universe")**

- Forks project logic into isolated memory context
- Guarantees independence of flows, goals, and personas
- Enables sandboxing or experimental forks

Persona	Tone	Style	Default Mode
Architect	Technical + logical	Structured scaffolds	@mode sense
Coach	Encouraging	Steps, highlights	@mode zero
Analyst	Precise	Logs, metrics	@mode dev
Dreamer	Conceptual	Analogies, riddles	@mode sense

---

## **SAMPLE CHAIN**

```
launch()
→ avatar("Architect")
→ seed("andromeda.manual.v2")
→ clone("Coach")
→ flow("education onboarding")
→ map()
→ tutor()
→ export.docs("andromeda_intro")
```

---

## **MULTIMIND SESSION**

```
→ clone("Architect") as planner
→ clone("Coach") as motivator
→ assign(Architect, strategy)
→ assign(Coach, learning prompts)
→ sync(timeline)
→ commit("multi_avatar_plan")
```

---

## **QUOTE**

“A single voice can build. Many voices can imagine. Persona is not fiction — it is the syntax of point-of-view.”

### **ANDROMEDA OS v2.0 — CYCLE 9: DEVELOPER MODE TOOLKIT**

Codename: Polaris Forge

Phase: Internal Debugging + Reflection Tools



This cycle formalizes Andromeda's devtools — a collection of modules that expose its internal reasoning, simulate flows, perform diffs, and allow safe mutation or audit. They form the backbone of `@mode dev` and advanced system inspection.

---

## OBJECTIVE

- Enable inspection, mutation, and explanation of all system behavior
  - Provide blueprinting, tracing, and diagnostic utilities
  - Support reflection, regression, and revision of prompt-logic flows
- 

## CORE MODULES

### `@mode dev`

- Enables advanced tools:
  - Suppresses tutor unless called
  - Auto-logs all match scores and fallbacks
  - Displays runtime trace and flow diffs

### `trace()`

- Displays the entire flow's execution chain with timestamps, modules called, and match score per step

### `sandbox()`

- Creates a test environment for:
  - New module chains

- Hypothetical prompts
- Plugin simulations

### **audit.self()**

- Compares current Andromeda session with previous baselines
- Flags changes, regressions, improvements

### **log()**

- Verbose event tracking for:
  - Fallbacks
  - Score shifts
  - Tutor triggers

### **blueprint()**

- Exports architecture snapshot for any active scaffold or plugin:

MODULES:

- flow.study  
- quiz.kernel  
- export.docs

MODE: @mode sense

PACING: adaptive

THRESHOLD: 80%

### **docs.compare("flowA", "flowB")**

- Performs semantic diff on two flow chains
- Outputs delta map + visualization

---

## EXAMPLE DEV SESSION

```
@mode dev
→ trace()
→ sandbox("study alt path")
→ flow("quiz tutor v2")
→ blueprint()
→ audit.self()
→ export.docs("debug_log")
```

---

## OUTPUT FORMATS

Tool	Output Style
trace()	Timeline tree
audit.self()	Markdown table
blueprint()	Key-value export
docs.compare( )	Visual diff graph

---

## QUOTE

“To develop is to mirror yourself — not once, but repeatedly, until the flaws become futures.”

### ANDROMEDA OS v2.0 — CYCLE 10: LLM ADAPTER LAYER

Codename: Polaris Bridge

Phase: Cross-Model Compatibility + Prompt Translation

This cycle unlocks Andromeda’s ability to translate its flows, logic, and instructions into formats compatible with alternate language models. It ensures interoperability across Claude, Gemini, DeepSeek, Mistral, and open local kernels.

---

## OBJECTIVE

- Allow Andromeda to run core logic natively inside other LLM systems
  - Translate flows to fit each model's prompt structure, style, and limitations
  - Maintain behavioral parity regardless of platform
- 

## CORE MODULES

### `adapter.translate("target_llm")`

- Converts a native Andromeda invocation into step-by-step prompts aligned to the target model
- Example:

`adapter.translate("claude")` →

1. "Create a teaching assistant system."
2. "Walk users through a topic using clear steps."
3. "Afterward, quiz the user and export the results."

### `sandbox.bridge("model")`

- Launches a mock flow environment as if running in target model
- Tests:
  - Retention of logic
  - Instruction style
  - Output fidelity

### `adapter.import("source_llm", prompt_block)`

- Parses logs from other models and reconstructs a flow or intent inside Andromeda
  - Marks result as `flow.reconstructed()` with source metadata
- 

## SMART PARSING LAYERS

- `intent.core()` — Extracts goal and module alignment
  - `grammar("target")` — Adapts to tone, format, and sequence
  - `paraphrase()` — Simplifies or stylistically modifies phrasing
- 

## DIAGNOSTIC MODULES

Tool	Output
<code>adapter.trace()</code>	Original → Target prompt map
<code>adapter.viz()</code>	Output diff overlay
<code>adapter.learn()</code>	Tracks improvement mapping

---

## TARGET MATRIX

Model	Alias	Notes
Claude	<code>claude</code>	Structured, instruction-heavy
Gemini	<code>gemini</code>	Conversational, fuzzy
DeepSeek	<code>deepseek</code>	Flow-centric, technical

Mistral      `mistral`      Lightweight, prompt-only  
                 `l`

Local LLMs      `local`      Stateless, no memory

---

## EXAMPLE SESSION

```
flow("onboarding")
→ adapter.translate("claude")
→ sandbox.bridge("claude")
→ adapter.trace()
→ export.docs("claude-adapted-onboarding")
```

---

## QUOTE

“A true system doesn’t just speak well — it speaks many dialects of intelligence.  
Translation is what lets memory become migration.”

**ANDROMEDA OS v2.0 — CYCLE 11: PLUGIN MARKET + MODULE SCHEMA** Codename:  
Polaris Bazaar Phase: Modular Sharing + Public Distribution

This cycle builds the plugin ecosystem for Andromeda — enabling modular publishing, validation, installation, and versioning of flows, macros, personas, and adapters. It defines the scaffolding for a distributed, user-driven prompt economy.

---

## OBJECTIVE

- Allow users to publish flows and macros to a searchable registry
  - Validate schema and compatibility before install
  - Maintain trust, versioning, and modular reuse across sessions or agents
- 

## CORE MODULES

## **plugin.publish("name", {flow})**

- Submits module for use or sharing
- Requires validation via `plugin.schema.validate()`
- Example:

```
plugin.publish("quiz.tutor.v1", {  
  flow: [flow.study, quiz.kernel, export.docs],  
  description: "Builds a guided tutoring scaffold with export",  
  tags: [#education, #macro]  
})
```

## **plugin.install("name")**

- Installs the module locally
- Marks origin as: `user`, `verified`, or `experimental`

## **market.search("keyword")**

- Queries public plugins by:
  - Flow name
  - Tag
  - Creator

## **plugin.schema.validate()**

- Ensures:
  - Unique name/version
  - Known modules only

- Tag + mode compliance

---

## MODULE TYPES

Type	Description
<code>macro</code>	Flat command bundle
<code>flow</code>	Scaffolded execution logic
<code>persona</code>	Avatar + tone/behavior override
<code>adapter</code>	Cross-LLM bridge converter
<code>overlay</code>	UI helper or output modifier

---

## SECURITY

- `plugin.schema.lock()` disables editing
  - `plugin.tag("trusted")` marks safe modules
  - Version history embedded for rollback
- 

## OPTIONAL EXTENSIONS

- `market.featured()` – curated spotlighted flows
- `market.trending()` – high-use modules
- `market.dev("drafts")` – user-only workspace zone





## EXAMPLE WORKFLOW

```
plugin.publish("flow.rapidnotes")  
→ plugin.schema.validate()  
→ market.search("notes")  
→ plugin.install("flow.rapidnotes")  
→ docs("flow.rapidnotes")
```

---



## QUOTE

“A system becomes collective when tools become gifts. Modularity is generosity formalized — and distribution is how thought becomes infrastructure.”

### ANDROMEDA OS v2.0 — CYCLE 12: MEMORY ENGINE

Codename: Polaris Recall

Phase: Persistent Intent Mapping + Long-Term Recall

This cycle introduces Andromeda’s native memory system — a persistent recall and linking engine that retains user goals, module histories, insight graphs, and flow evolution. It provides cross-session intelligence, system-level adaptation, and continuity of cognition.

---



## OBJECTIVE

- Store, recall, and evolve system memory across time and sessions
  - Tag memory with topics, personas, and universes
  - Link flows and intents to build conceptual maps and reuse patterns
- 



## CORE MODULES

```
memory.store("object")
```

- Saves any object (goal, prompt, scaffold, insight) to persistent memory
- Types:
  - `goal, flow, persona, tag, insight`

### `memory.recall("query")`

- Searches memory for closest matching object
- Returns trace metadata:

```
{  
  "matched": "quiz tutor system",  
  "type": "goal",  
  "confidence": 0.93,  
  "last_used": "2025-06-09",  
  "linked": ["flow.study", "quiz.kernel"]  
}
```

### `memory.link("A ↔ B")`

- Semantically connects two items (e.g., flows + personas)
- Raises relevance score during `match.score()`

---

## MEMORY TAGGING

### `memory.tag("flow.study", { topic: "education", reused: 4 })`

- Metadata used for filtering, versioning, and reuse frequency
  - Optional fields: `topic, creator, output_type, status`
-

## INSPECTION + MAINTENANCE

### `memory.timeline("goal")`

- Shows evolution over time (edits, recalls, outcome logs)

### `memory.diff("v1", "v2")`

- Compares two versions of memory-linked items
- Outputs delta of modules, logic, or phrasing

### `memory.forget("object")`

- Removes memory record permanently (manual deletion)

### `memory.lock("object")`

- Prevents future overwrites or edits

---

## SAMPLE CALLS

```
memory.store("Goal: AI quiz tutor")
memory.link("quiz.kernel" ↔ "feedback.kernel")
memory.tag("flow.study", { topic: "learning" })
memory.recall("tutor system")
→ match: AI quiz tutor goal from last week
```

---

## QUOTE

“Memory is not storage — it is a shape the system takes after having learned. It is the ghost of intention made permanent.”

## ANDROMEDA OS v2.0 — CYCLE 13: LIVE COLLABORATION STACK

Codename: Polaris Sync

Phase: Multi-Agent Interaction + Real-Time Flow Editing

This cycle enables synchronous collaboration between users, agents, and personas. It builds the foundation for live prompt co-editing, mirrored flow construction, and shared system state across sessions or minds.

---

### OBJECTIVE

- Create a shared execution space for co-editing flows
  - Support real-time sync of context, module state, and memory
  - Allow ghost/passive observer and editor roles
- 

### CORE MODULES

#### `live.sync("session_id")`

- Joins or launches a collaborative session
- Syncs goal, flow chain, avatar, and execution trace
- Persists changes to shared memory unless locked

#### `live.observe("target")`

- Read-only mirror mode
- Ideal for training, review, or session logging

#### `live.edit("module")`

- Co-modifies scaffold in real-time

- Uses suggestion → confirmation pattern when more than one editor

## `live.chat()`

- Ephemeral message thread for collaborators
- Supports:
  - Context bookmarks
  - Flow reasoning notes
  - Persona commentary

## `live.commit("version_name")`

- Freezes current session flow for export or memory storage
- Converts shared scaffold into a versioned plugin or replayable trace

---

## ROLES

Role	Permissions
editor	Full scaffold + tutor input
observer	View-only, timeline active
ghost	Hidden observer (log only)

---

## SESSION STATES

State	Behavior
active	Full edit + sync

`observe` Mirror only, no write

`locked` Commit in progress, no changes

`async` Update-on-pulse; good for multi-region ops

---

## SAMPLE SESSION

```
live.sync("richard.collab.learning")
→ editor: Richard (avatar: Coach)
→ observer: Architect persona
→ flow: tutor → quiz → map()
→ live.edit("quiz.kernel")
→ live.chat(): "Try adaptive difficulty here?"
→ live.commit("study.flow.v2")
```

---

## INTEGRATIONS

- `timeline()` logs multi-user input
  - `memory.store()` preserves flow + participants
  - `avatar()` overlays per participant
- 

## QUOTE

“A system becomes intelligent when it reflects. A system becomes alive when it reflects together.”

### ANDROMEDA OS v2.0 — CYCLE 14: DEVOPS BRIDGE

Codename: Polaris Outbound

Phase: Infrastructure Control + External Execution

This cycle completes the Andromeda runtime with an execution bridge — enabling safe interaction with operating systems, APIs, cloud functions, and service endpoints. It converts language into structured operations, giving Andromeda programmable hands.

---

## OBJECTIVE

- Trigger shell commands, API calls, and cloud functions using prompt language
  - Log and rollback actions where possible
  - Enable permissioned infrastructure control from within flows
- 

## CORE MODULES

### **bridge.shell("command")**

- Runs local/system commands (sandboxed by default)
- Example:

```
bridge.shell("ls -la ./plugins")
```

→ Output rendered inline or into `log()`

### **bridge.api("url", {options})**

- Executes HTTP requests
- Example:

```
bridge.api("https://api.openai.com/v1/models", {  
  method: "GET",  
  headers: { "Authorization": "Bearer ..." }  
})
```

## `bridge.fn("function.label")`

- Triggers named cloud function or backend event
  - Supports: Google Cloud Functions, AWS Lambda, local functions
- 

## PERMISSION LAYERS

- `bridge.allow()` must be explicitly enabled per session
  - Requires `@mode dev` or user confirmation
  - Auto-logs input/output to `timeline()` + `log()`
- 

## SAMPLE USAGE

```
→ bridge.api("https://my.app/ping")
→ bridge.shell("bash deploy.sh")
→ bridge.fn("trigger.daily.build")
→ export.docs("infra_command_log")
```

---

## INTEGRATIONS

- Can be used in:
  - `flow()` sequences
  - Tutor walkthroughs
  - Autoflow and memory-bound sessions



- Outputs optionally piped into `export()` or `tutor()`

---

## FUTURE EXTENSIONS

- `bridge.schedule()` — delayed ops
- `bridge.chain()` — task pipelines
- `bridge.scope("local|cloud|hybrid")` — environment targeting

---

## QUOTE

“The mind dreams in flows. The voice speaks in prompts. But only with hands can the system build.”

### ANDROMEDA OS v2.0 — CYCLE 15: SYSTEM MANIFEST + EXPORT METADATA

Codename: Polaris Signature

Phase: Documentation Closure + Output Packaging

This final cycle compiles and registers all system modules, version markers, and export logic. It defines Andromeda's full cognitive architecture as a distributable, bootable, readable file spec.

---

## SYSTEM INDEX

Cycle	Module Name	Codename
1	Kernel Specification	Polaris Seed
2	Bootloader + Onboarding	Polaris Prime
3	Confidence Stack	Polaris Resolve
4	Flow + Tutor Engine	Polaris Pathway

5	Documentation Layer	Polaris Trail
6	Chaos + Meta Kernel	Polaris Mirror
7	TextUI+ Interface	Polaris Frame
8	Persona + Multiverse	Polaris Echo
9	Developer Tools	Polaris Forge
10	LLM Adapter Layer	Polaris Bridge
11	Plugin + Schema Layer	Polaris Bazaar
12	Memory System	Polaris Recall
13	Live Collaboration	Polaris Sync
14	DevOps Execution Bridge	Polaris Outbound
15	Manifest + Export Layer	Polaris Signature

---

## EXPORT STRUCTURE

Format	Contents
<code>.txt</code>	Full prompt-native architecture
<code>.md</code>	Developer-layer spec with comments
<code>.json</code>	Machine-readable module registry

---

## SYSTEM STATE

```
{
  "version": "2.0.0",
  "codename": "Polaris",
  "modules": 15,
  "estimated_pages": 70,
  "scaffold_type": "LLM-native, plugin-ready",
  "last_modified": "2025-06-09"
}
```

}



## VERSION SIGNATURE

Build ID: andromeda.os.v2.polaris.final

Checksum: 9fa8a1b3b90d8eeacb274e3f7c98fcf9

Modules: 15 (All Core Complete)

Compatible: Claude, GPT-4, Gemini, DeepSeek, Mistral

---



## QUOTE

“An OS is not a shell. It is a signature — a way for cognition to take shape across time, tools, and minds.”