mshell Ecosystem — Tools & Components Reference that relates to workflow inside it.

Overview

mshell is a polyglot AI-native shell environment developed by Art2Dec SoftLab. It provides a unified execution platform where code in multiple programming languages, AI model calls, and data pipelines are defined in a single Markdown document and executed as a coherent workflow. The system is designed around a simple but powerful idea: **variables as files, language blocks as pipeline stages, and LLM models as first-class citizens of the execution graph**.

The mshell ecosystem consists of four main tools that work together:

Components

---

1. mshell — The Core Engine

mshell is the heart of the system. It is a command-line shell that extends traditional shell capabilities with native polyglot execution and AI model integration.

What it does:

mshell parses and executes Markdown documents containing code blocks in any supported language. It manages the inter-language communication through a **file-based context system** — each named variable is stored as a file in a session-specific directory (`/tmp/mshell_ctx_PID/`). All languages and tools in the pipeline share this context, enabling seamless data flow between, for example, a C block that computes numbers and a Python block that analyzes them, or a Bash block that prepares a prompt and an LLM directive that processes it.

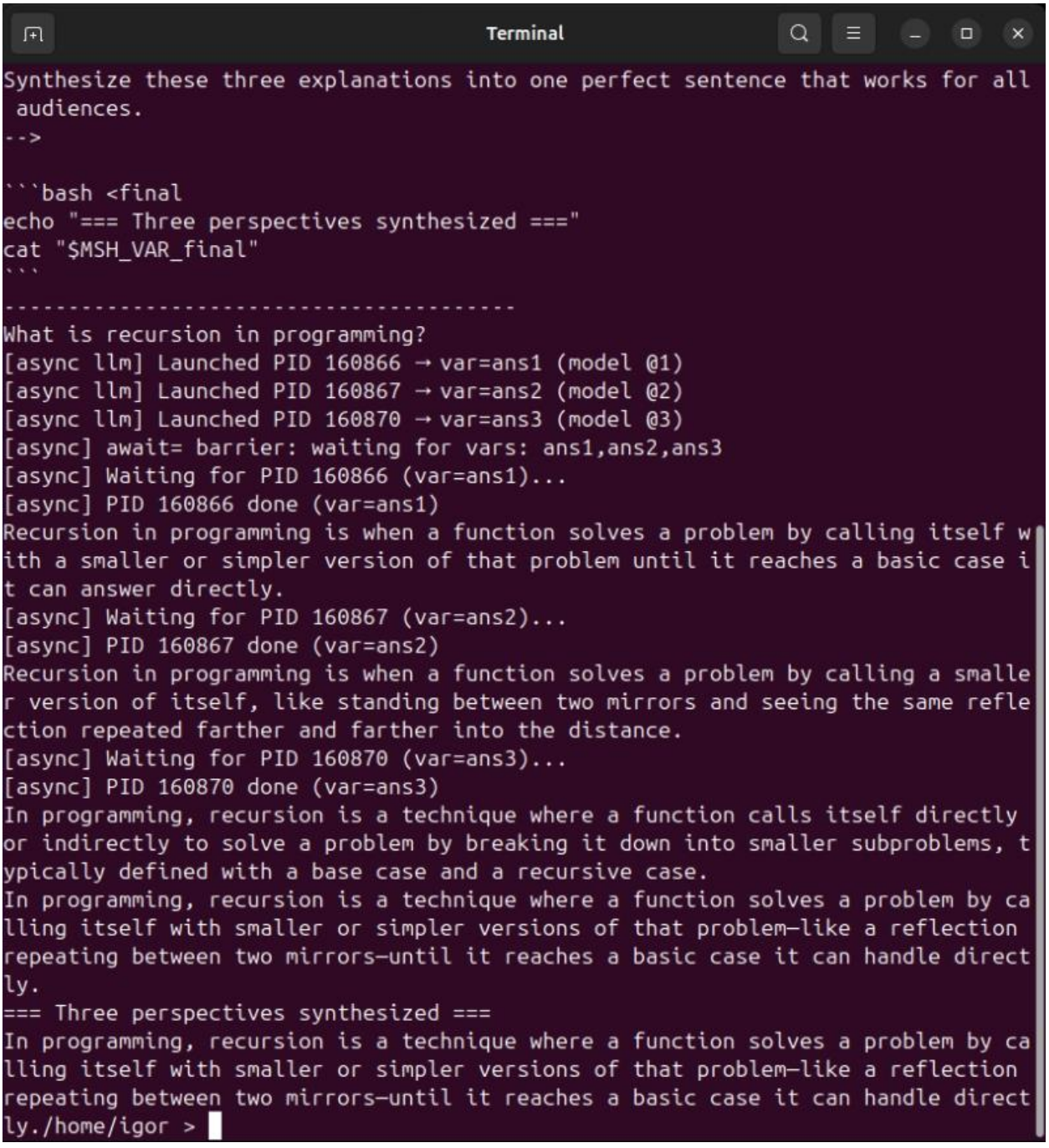
Supported languages: Bash, Python, C, C++, Rust, Go, Lua, MShell (native)

AI model integration: Up to 3 LLM models (`@1`, `@2`, `@3`) configurable independently — local models via Ollama or remote APIs (OpenAI, Anthropic Claude, and others). Models can be called synchronously or launched as async background processes with an await barrier for synchronization.

Key execution features: - Sequential top-to-bottom pipeline execution - Conditional block execution (`if=var:value`) - Loop control (`<!--@loop max=N until=var:value-->`) - Async

parallel LLM execution with `await` barriers - LLM exec mode — model generates code which is immediately compiled and executed - LLM orchestrator mode — model generates a complete Markdown workflow which is recursively executed - Variable passing between all languages via `>outvar` / `<invar` fence attributes - Native mshell commands: `print`, `eval`, `readfile`, `ollama1/2/3`, and more

Interactive mode: mshell also works as an interactive shell for direct command entry — variables set interactively persist in the same context and are accessible to subsequently executed Markdown blocks.



2. mshell-flow — The Visual Workflow Designer

mshell-flow is a GTK-based graphical workflow editor that provides a visual representation of mshell Markdown pipelines. It allows users to design, inspect, edit, and export workflows as diagrams without manually writing Markdown syntax.

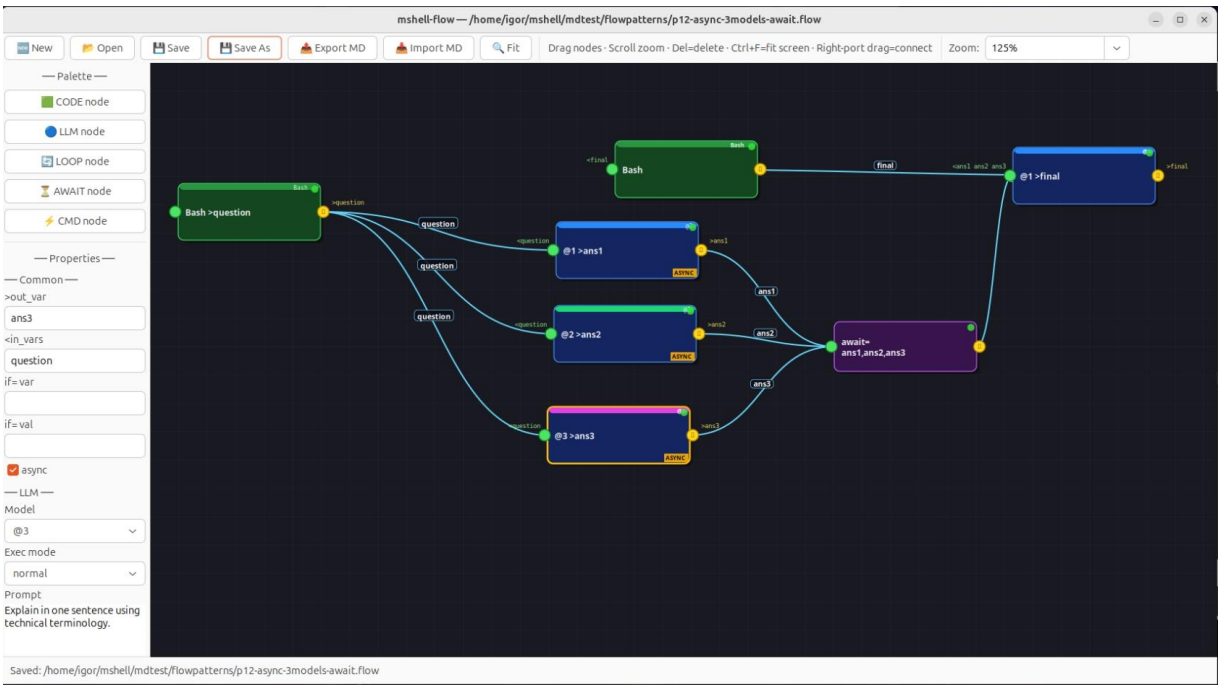
What it does:

Each node in the diagram corresponds to a block in the Markdown pipeline. Edges between nodes represent variable dependencies — an edge labeled `topic` means the source node writes `>topic` and the destination node reads `<topic`. The diagram is a direct visual representation of the underlying Markdown document.

Node types: - **CODE node** — a code block in any of the 7 supported languages (Bash, Python, C, C++, Rust, Go, Lua, MShell). Configurable language, code body, input variables, output variable, and conditional execution - **LLM node** — an AI model directive (`<!--@N -->`). Configurable model number (1–3), exec mode (normal / exec / orchestrator), async flag, prompt text, input variables, output variable, and conditional execution - **LOOP node** — defines a loop region with maximum iteration count and exit condition (`until=var:value`) - **AWAIT node** — a synchronization barrier that waits for a set of async variables to be written before proceeding - **CMD node** — a native mshell command block (multiline mshell commands in a fenced block)

Key features: - **Import MD** — parse an existing Markdown workflow document and automatically reconstruct the diagram with nodes and edges - **Export MD** — serialize the current diagram back to a valid Markdown document ready for execution in mshell - **Visual variable routing** — edges are automatically labeled with variable names; port indicators show which variables flow in and out of each node - **Async indicators** — LLM nodes marked as async display an ASYNC badge - **Properties panel** — clicking any node opens its full property set for editing (code, prompt, model, variables, conditions) - **Zoom, pan, fit** — standard canvas navigation; Ctrl+F fits all nodes on screen - **Save/Load** — workflows saved as `.flow` JSON files preserving node positions and all properties

Workflow: Design visually → Export MD → execute in mshell. Or: write Markdown → Import MD → adjust layout → Save as .flow for reuse.



---

3. mshell Editor (edi) — The Markdown Workflow Editor

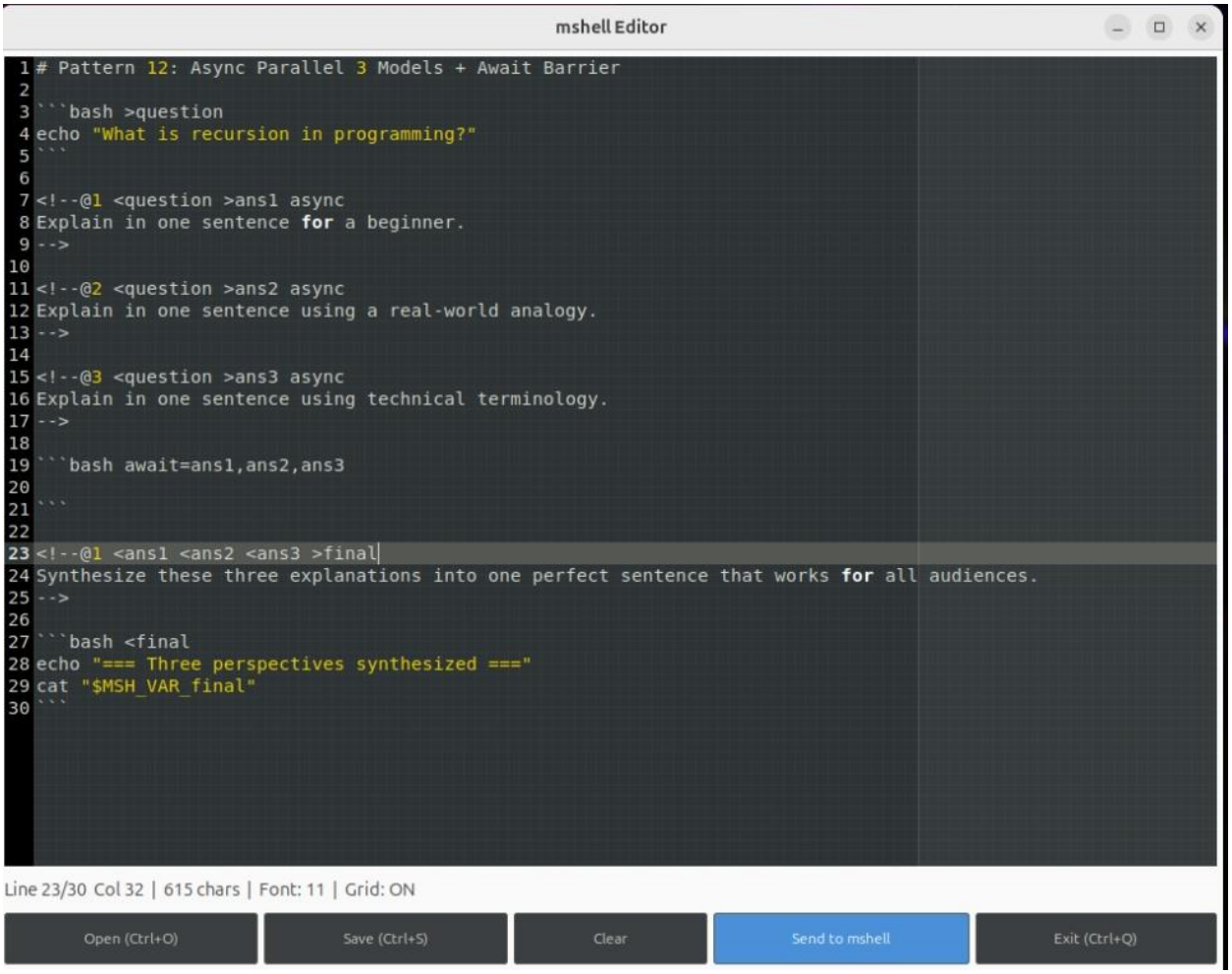
edi is a lightweight GTK-based text editor specifically designed for authoring and editing mshell Markdown workflow documents. It integrates directly with the running mshell session.

What it does:

edi provides a syntax-aware editing environment for mshell Markdown files. It communicates with a running mshell instance via a named pipe, allowing the user to send the current document for immediate execution without leaving the editor.

Key features: - Syntax highlighting for mshell Markdown — code fences, LLM directives, variable declarations, and mshell commands are visually distinguished - **Send to mshell** — one-click execution of the current document in the running mshell session via named pipe - Open / Save / Save As — standard file operations for `.md` workflow files - Line and column indicator, character count - Adjustable font size - Works alongside mshell — the editor and shell run concurrently; the user can continue using the shell interactively while editing

Typical use: Open a workflow Markdown file, edit prompts or code, press “Send to mshell” — the document is immediately executed and results appear in the terminal. Iterate rapidly without leaving the tool.



---

4. Futuristic Notebook — The Documentation Tool

The Notebook is a multi-page GTK document editor designed for creating rich documentation from Markdown content. It serves as the presentation and knowledge-management layer of the mshell ecosystem.

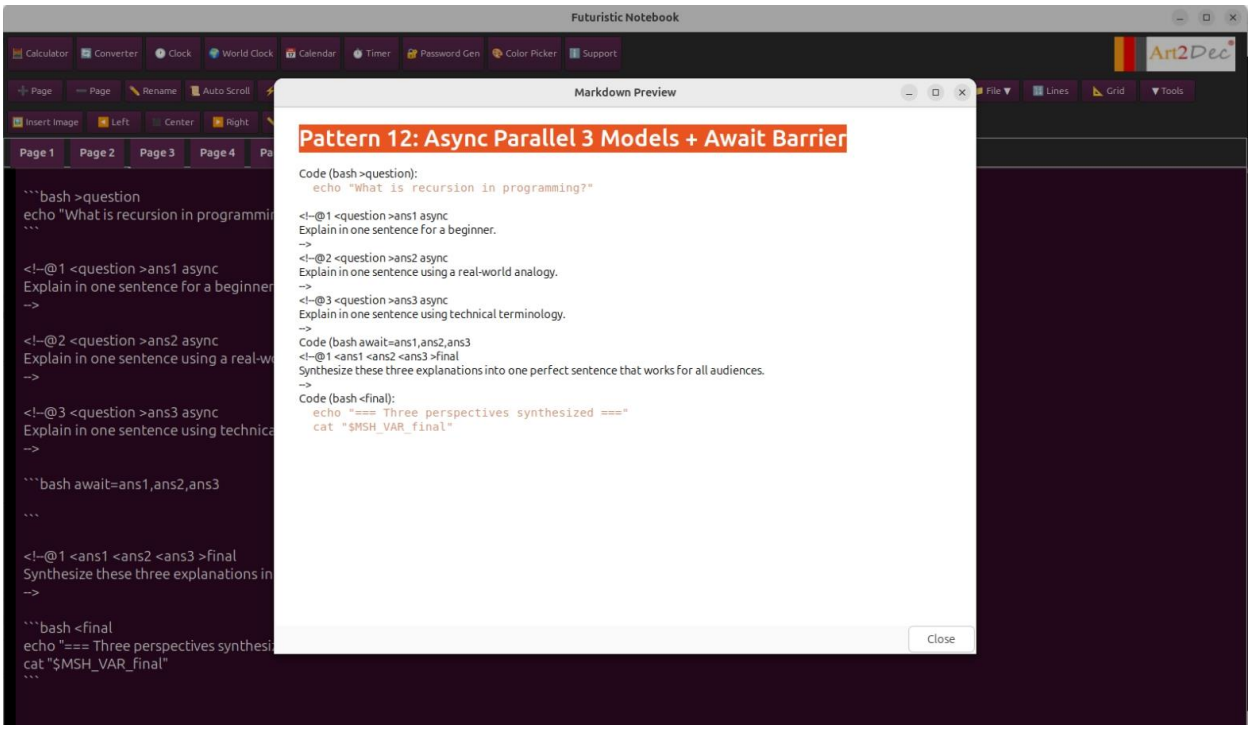
What it does:

The Notebook provides a paginated document environment where mshell Markdown files can be imported, rendered, and organized into multi-page documents. It is designed for creating structured documentation, user manuals, pattern libraries, and reports from workflow results.

Key features: - Multi-page document structure with named tabs - **Markdown Preview** — renders the current page as formatted HTML showing headings, code blocks with syntax coloring, and inline formatting - Import Markdown files directly into pages - Insert images into pages - Text alignment controls (left, center, right) - Auto Scroll for long pages - Export document — save as file for distribution - Built-in utility tools accessible from the toolbar:

Calculator, Converter, Clock, World Clock, Calendar, Timer, Password Generator, Color Picker - Grid and line guides for layout alignment

Typical use: After developing and testing a set of workflow patterns in mshell and mshell-flow, the Notebook is used to assemble them into a structured reference document — importing the Markdown descriptions, adding diagrams and screenshots, and exporting the result as a distributable document.



How the Tools Work Together

```
mshell-flow          →   Export MD   →   mshell (executes pipeline)
    ↑                                          ↓
Import MD            ←   edi (edit)   ←   results / iteration

Notebook  ←  Import pattern MDs  ←  mshell-flow Export MD
```

A typical development cycle:

1. **Design** the pipeline visually in **mshell-flow**, placing CODE and LLM nodes and connecting them with variable edges
2. **Export MD** to generate the Markdown document
3. **Open in edi** to fine-tune prompts, code, and variable names
4. **Send to mshell** from edi for immediate execution
5. Observe results in the terminal, iterate in edi or mshell-flow
6. **Save as .flow** in mshell-flow for reuse

7. **Document** the final patterns in the **Notebook**, assembling them into a reference guide with Markdown preview

---

Advantages of the mshell Ecosystem

Unified polyglot execution

Most workflow tools force you to choose one language or use awkward bridging mechanisms. mshell treats 7 languages and 3 AI models as equal pipeline citizens — a C block, a Python block, and an LLM call are all just steps in the same document, sharing the same variable context.

Markdown as the execution format

The pipeline definition is human-readable, version-controllable plain text. There is no proprietary binary format, no GUI-only workflow, no lock-in. Any text editor can author mshell workflows; the Markdown files are also valid documentation.

Visual and textual representations are equivalent

mshell-flow diagrams and Markdown documents are two representations of the same thing. Import MD builds the diagram; Export MD produces valid executable Markdown. Neither representation is "the real one" — they are synchronized views.

AI models as first-class pipeline stages

LLM calls are not external API wrappers bolted onto a workflow system — they are native pipeline stages with the same variable input/output mechanism as code blocks. An LLM can receive structured data from a C computation, process it, and pass the result to a Rust formatter without any glue code.

Async parallelism with simple syntax

Launching three LLM models in parallel requires only adding `async` to each directive. The `await` barrier waits for all of them. No threads, no callbacks, no futures — the complexity is handled by the engine.

Loop and conditional constructs

`<!--@loop-->` and `if=var:value` bring control flow to the pipeline level. An evaluator-optimizer loop that runs until a quality threshold is met — or a router that sends tasks to different handlers based on LLM classification — requires no custom code, only standard mshell syntax.

Comparison with Existing Tools

vs. LangChain / LlamaIndex

LangChain and LlamaIndex are Python frameworks for building LLM-powered applications. They require Python code for every step, have no native multi-language support, and provide no visual workflow editor. mshell supports 7 languages natively, requires no framework code for simple pipelines, and provides a visual designer. LangChain is more mature with a larger ecosystem; mshell is more accessible for rapid prototyping and multi-language workflows.

vs. n8n / Zapier / Make

These are visual no-code/low-code automation platforms. They focus on connecting SaaS services and APIs through a GUI. They are not designed for code execution, compiled languages, or local LLM models. mshell is a developer tool for technical pipelines involving real code compilation and execution, not service integration.

vs. Apache Airflow / Prefect

Airflow and Prefect are workflow orchestration systems for data engineering at scale. They require Python DAG definitions, infrastructure setup, and are designed for production data pipelines with scheduling, retries, and monitoring. mshell is designed for interactive development, rapid experimentation, and local execution — not for production orchestration at scale.

vs. Jupyter Notebooks

Jupyter provides interactive polyglot execution via kernels, but each cell runs in isolation per kernel — there is no native inter-cell variable passing between different languages. Connecting a Python cell to a Rust cell requires manual file I/O. mshell's variable system handles this transparently. Jupyter has no native LLM integration and no visual workflow editor.

vs. GitHub Actions / GitLab CI

These are CI/CD pipeline systems. They support multi-language jobs connected through artifact passing, but they are designed for automated build/test/deploy workflows, not for interactive AI-driven data pipelines. They have no LLM integration and no visual designer.

Uniqueness

mshell occupies a niche that no existing tool covers completely:

A locally-executed, visually-designed, polyglot, AI-native workflow shell with a unified variable context.

The combination of: - 7 compiled and interpreted languages in one execution context - Up to 3 configurable LLM models as native pipeline stages - A visual designer that round-trips to executable Markdown - A file-based variable system accessible to all languages simultaneously - Loop, conditional, and async constructs at the pipeline level - An integrated editor with direct shell communication - A documentation tool for assembling workflow libraries

…does not exist as a single integrated system in any other tool at the time of writing.

mshell is particularly well-suited for: - AI-assisted data processing pipelines combining computation and natural language reasoning - Rapid prototyping of multi-model LLM workflows - Educational demonstrations of polyglot programming and agentic AI patterns - Building self-improving code generation pipelines (generate → review → improve loops) - Research workflows requiring both numerical computation and language model analysis

---

Created by Igor Lukyanov, Art2Dec SoftLab (non-profit software laboratory), February 25th, 2026.

Mshell v.1.4.1 cheatsheet:
https://www.appservgrid.com/paw92/index.php/2026/02/04/mshell-v-1-4-1-cheatsheet-january-26th-2026/

Permanent link to workflow patterns reference quide :
https://www.appservgrid.com/paw92/index.php/2026/02/26/mshell-workflow-patterns-reference-guide-part-i-p1-p13/

Permanent link to workflow quide :
https://www.appservgrid.com/paw92/index.php/2026/02/23/mshell-workflow-guide-february-22nd-2026/