MOL

The Cognitive Programming Language

Native pipeline operators | Auto-tracing | AI domain types | RAG built-in

v0.3.0 68 Tests Passing 90+ Stdlib Functions Python 3.10+

What is MOL?

MOL (Mind-Oriented Language) is the **first programming language** with native pipeline operators and automatic execution tracing — purpose-built for AI/RAG pipelines, cognitive computing, and data processing. Created by **Mounesh Kodi** for **IntraMind** at **CruxLabx**.

```
-- A full RAG pipeline in ONE expression
 let doc be Document("notes.txt", "MOL is built for IntraMind.")
 doc |> chunk(512) |> embed |> store("index")
-- Every step auto-traced: timing, types, values
```

90+	8	33	68	2
Stdlib Functions	Domain Types	AST Nodes	Tests Passing	Transpile Targets

Why MOL?

Problem	Python / JS	MOL
Pipeline Debugging	`print()` everywhere	`|>` auto-traces every step
Data Flow	No pipe operator	`|>` left-to-right flow
AI Type Safety	Generic dicts	Native `Thought`, `Document`, `Embedding`
RAG Boilerplate	50+ lines setup	One expression
Safety Rails	Hope for the best	`guard` + `access` control
Portability	Rewrite per language	Transpiles to Python & JS

Key Features

Pipeline Operator |>

- Data flows left → right through functions
- Auto-tracing at 3+ stages (timing, types)
- Named pipelines with `pipeline` keyword
- User functions as pipe stages

Domain Types

- **Core:** Thought, Memory, Node, Stream
- **RAG:** Document, Chunk, Embedding, VectorStore
- Field access, methods, constructors
- Type annotations with enforcement

90+ Standard Library

- Functional: map, filter, reduce, flatten, zip
- Math/Stats: mean, median, stdev, sin, cos, log
- Strings: format, pad, starts_with, index_of
- Hashing: SHA-256, UUID, Base64
- Random, sorting, binary search

Safety & Tooling

- `guard` assertions with messages
- Access control (SecurityContext)
- CLI: run, parse, transpile, REPL
- VS Code extension (syntax, snippets)
- Transpiles to Python & JavaScript

Language at a Glance

Variables & Types

```
let name be "IntraMind"
let score : Number be 42
let items be [1, 2, 3]
let config be {"key": "value"}
set score to score + 1
```

Functions & Pipelines

```
define greet(name)
  return "Hello, " + name + "!"
end

pipeline preprocess(data)
  return data |> trim |> lower
end

-- First-class functions
let doubled be map([1,2,3], double)
```

Control Flow

```
if score > 90 then
  show "excellent"
elif score > 70 then
  show "good"
else
  show "needs work"
end

for item in range(5) do
  show to_text(item)
end
```

Guards & Safety

```
guard confidence > 0.8 : "Too low"
guard len(data) > 0 : "Empty"
access "mind_core"
-- Throws MOLSecurityError if denied
```

Standard Library (90+ Functions) & Domain Types

Category	Functions
General	len, type_of, to_text, to_number, range, abs, round, sqrt, max, min, sum, print
Functional	map, filter, reduce, flatten, unique, zip, enumerate, count, find, find_index, take, drop, group_by, chunk_list, every, some
Math	floor, ceil, log, sin, cos, tan, pi, e, pow, clamp, lerp
Statistics	mean, median, stdev, variance, percentile
Collections	sort, sort_by, sort_desc, binary_search, reverse, push, pop, keys, values, contains, join, slice
Strings	split, upper, lower, trim, replace, starts_with, ends_with, pad_left, pad_right, repeat, char_at, index_of, format
Hash / Encode	hash (SHA-256/MD5/SHA-1/SHA-512), uuid, base64_encode, base64_decode
Random	random, random_int, shuffle, sample, choice
Map Utils	merge, pick, omit | **Type Checks:** is_null, is_number, is_text, is_list, is_map
RAG Pipeline	load_text, chunk, embed, store, retrieve, cosine_sim, think, recall, classify, summarize
Debug	display, tap, assert_min, assert_not_null, inspect, to_json, from_json, clock, wait

Type	Category	Purpose	Constructor
Thought	Core	Cognitive unit with confidence	`Thought("idea", 0.9)`
Memory	Core	Key-value with decay	`Memory("key", value)`
Node	Core	Neural graph vertex	`Node("label", 0.5)`
Stream	Core	Real-time data buffer	`Stream("feed")`
Document	RAG	Text with source metadata	`Document("file", "text")`
Chunk	RAG	Text fragment	`Chunk("text", 0, "src")`
Embedding	RAG	64-dim vector (deterministic)	`Embedding("text", "model")`
VectorStore	RAG	In-memory vector index	Created via `store()`

CLI & Architecture

CLI Commands

```
mol run file.mol
mol run file.mol --no-trace
mol parse file.mol
mol transpile file.mol -t python
mol transpile file.mol -t js
mol repl
mol version
```

Architecture

$$\begin{tikzcd}
\texttt{.mol} \arrow[r, "\text{Lark}"] & \texttt{AST} \arrow[r, "\text{Visitor}"] \arrow[dr] & \texttt{Execute} \\
 & & \texttt{Transpile} \arrow[r] & \texttt{Py / JS}
\end{tikzcd}$$

Stack: Python 3.10+ | Lark 1.3.1
Parser: LALR → 33 AST node types
Interp: Tree-walking + closures
Extension: VS Code (TextMate + snippets)

Version History & Roadmap

Version	Date	Status	Highlights
v0.1.0	2026-02-08	Done	Grammar, AST, interpreter, 4 domain types, CLI, transpiler, 30+ stdlib
v0.2.0	2026-02-09	Done	Pipeline `|>`, auto-tracing, guard, 4 RAG types, 15 RAG functions
v0.3.0	2026-02-10	Done	42 new algorithms, 90+ stdlib, callable functions, documentation
v0.4.0	—	Next	Sovereign AI — agent blocks, model registry, knowledge graph
v0.5.0	—	Planned	Async pipelines, real DB integration (FAISS/Qdrant), HTTP server
v1.0.0	—	Vision	Package manager, playground, debugger, cloud deployment

---

Built for IntraMind by CruxLabx

Creator: Mounesh Kodi · https://github.com/crux-ecosystem/mol-lang