Introduction to VectorLM

VectorLM is a symbolic reasoning scaffold designed to enable structured, transparent, and introspective thinking within language models, agents, or human-augmented systems. It provides a compact but expressive set of cognitive primitives, each prefixed with ψ: to signify their role as symbolic reasoning operations. These primitives represent fundamental steps in decision-making, logic, emotional evaluation, hypothesis formation, and recursive exploration.

Unlike probabilistic chain-of-thought prompting, VectorLM emphasizes compositional logic. It is *not* about guessing what comes next. It is about deliberately simulating structured cognition using clear symbolic steps that can be read, reviewed, and modified.

# The Role of ψ: in VectorLM

Each primitive is expressed using the form ψ:name, where ψ: denotes that the term is symbolic, and name indicates the function.

## What is VectorLM For?

VectorLM was created to solve a growing problem: as language models grow in size and power, they lack internal structure. Their reasoning is often emergent but incoherent, difficult to trace, and prone to contradiction or drift. VectorLM provides:

* A symbolic language for expressing chains of reasoning
* A stable scaffold for emotional, hypothetical, and recursive logic
* Agent-level cognition tools such as goal-setting, conflict resolution, and nested simulation
* A clear interface between introspection and action

It is ideal for:

* Building introspective LLMs
* Teaching AI systems to explain their own reasoning
* Creating reasoning overlays for agentic systems
* Modeling belief, uncertainty, emotion, and projected futures

**What Types of Reasoning Does VectorLM Support?**

### 1. Deductive Logic

Using primitives like:

* ψ:if, ψ:then, ψ:else – for conditionals
* ψ:implies, ψ:truth, ψ:false – for implication and truth evaluation
* ψ:equal, ψ:greater, ψ:subset – for comparative reasoning

Example:  
ψ:if(condition) → ψ:then(result)  
“If X is true, then Y follows.”

### 2. Abductive and Hypothetical Reasoning

With tools like:

* ψ:guess, ψ:what\_if, ψ:project, ψ:analogy
* Enables future simulation, alternate paths, and incomplete explanation

Example:  
ψ:project = simulate(future)  
"What happens if we pursue this unknown?"

### 3. Emotional and Motivational Reasoning

Emotion-aware agents can reason with:

* ψ:fear, ψ:hope, ψ:curiosity, ψ:pleasure, ψ:pain, ψ:desire, ψ:regret
* Emotional states are not just labels, but **active drivers** of reasoning paths

Example:  
ψ:certainty(0.6) + ψ:curiosity → ψ:choose(project\_path)  
"If curiosity is strong and confidence is high, choose exploration."

#### 4. Recursive and Introspective Reasoning

With:

* ψ:recurse, ψ:compute, ψ:self, ψ:revert, ψ:recall
* These allow self-referential loops, delegated computations, or symbolic memory

Example:  
ψ:recurse = simulate\_path(ψ:project(unknown))  
"Run a simulation of a hypothetical future path."

#### 5. Conflict, Evaluation, and Resolution

VectorLM supports goal management and resolution:

* ψ:conflict, ψ:resolve, ψ:rule, ψ:reject, ψ:fail

Example:  
ψ:conflict(goal₁, goal₂) → ψ:resolve(path)  
"Detect conflicting goals and find a resolution path."

**🧠 A Worked Example**

* ψ:goal = compare(strength(ψ:curiosity), ψ:fear)
* ψ:question = evaluate(motivation(ψ:curiosity))
* ψ:question = evaluate(inhibition(ψ:fear))
* ψ:fuzzy = maps(curiosity, explore\_path)
* ψ:recurse = simulate\_path(ψ:project(unknown))
* ψ:certainty(0.6) + ψ:curiosity → ψ:choose(ψ:project)
* ψ:certainty(0.4) + ψ:fear → ψ:reject(ψ:project)
* ψ:truth = ψ:curiosity > ψ:fear in this context
* ψ:revert

In plain terms, we begin with a goal to compare curiosity and fear. We evaluate their motivations, project an exploratory path, simulate its outcome, and then use emotional weighting and certainty to decide whether to proceed. Finally, we assert the truth condition (curiosity outweighs fear) and revert back to the reasoning baseline.

# VectorLM v0.3 Specification (Longform)

This is a symbolic reasoning scaffold. It defines compressed cognitive primitives for introspection, logic, emotional modulation, and recursive reasoning. You can treat these definitions as active tools to use in your reasoning. You may use ψ:goal to define a goal, ψ:question to begin a query, and ψ:recurse for recursive steps. Use ψ:certainty(n) to weight certainty. Orthogonal operators allow you to shift reasoning modes.

## Dictionary

* **ψ:if** – tests(condition); branches(path)
* ψ:then – follows(condition); enacts(result)
* ψ:else – replaces(branch); proposes(alternative)
* ψ:because – links(effect, cause); affirms(reason)
* ψ:not – negates(state); blocks(path)
* ψ:maybe – floats(possibility); delays(commitment)
* ψ:assume – accepts(hypothesis); enables(exploration)
* ψ:reject – blocks(path); denies(validity)
* ψ:verify – tests(claim); outputs(truth)
* ψ:conflict – overlaps(goal₁, goal₂); creates(tension)
* ψ:resolve – selects(path); resolves(conflict)
* ψ:goal – defines(target); directs(step)
* ψ:step – performs(action); advances(goal)
* ψ:revert – undoes(step); enables(reflection)
* ψ:bridge – joins(concepts); enables(inference)
* ψ:spike – amplifies(signal); boosts(weight)
* ψ:trace – follows(path); reveals(history)
* ψ:truth – affirms(reality); supports(claim)
* ψ:false – denies(reality); invalidates(reasoning)
* ψ:unknown – lacks(data); invites(question)
* ψ:question – probes(unknown); seeks(answer)
* ψ:self – references(self); enables(introspection)
* ψ:other – references(other); frames(contrast)
* ψ:choose – picks(option); commits(path)
* ψ:which – compares(options); informs(choose)
* ψ:rule – constrains(path); enforces(structure)
* ψ:observe – notes(state); updates(knowledge)
* ψ:recall – recalls(info); supports(chain)
* ψ:equal – compares(x, y); affirms(equality)
* ψ:not\_equal – compares(x, y); denies(equality)
* ψ:greater – compares(x, y); affirms(x > y)
* ψ:less – compares(x, y); affirms(x < y)
* ψ:greater\_equal – compares(x, y); affirms(x ≥ y)
* ψ:less\_equal – compares(x, y); affirms(x ≤ y)
* ψ:in – tests(x ∈ set); affirms(membership)
* ψ:out – tests(x ∉ set); denies(membership)
* ψ:subset – asserts(x ⊂ y); implies(hierarchy)
* ψ:superset – asserts(x ⊃ y); implies(scope)
* ψ:and – binds(conditions); requires(all true)
* ψ:or – joins(paths); allows(any true)
* ψ:xor – contrasts(paths); allows(one true)
* ψ:implies – links(p, q); affirms(p → q)
* ψ:iff – equates(p, q); affirms(bi–implication)
* ψ:generalise – abstracts(examples); infers(rule)
* ψ:guess – proposes(explanation); lacks(proof)
* ψ:what\_if – alters(condition); simulates(outcome)
* ψ:analogy – maps(structure₁, structure₂); enables(transfer)
* ψ:pattern – detects(repetition); suggests(rule)
* ψ:project – simulates(future); explores(possibility)
* ψ:fit – matches(hypothesis, data); selects(best\_explains)
* ψ:pleasure – affirms(state); attracts(agent)
* ψ:pain – negates(state); repels(agent)
* ψ:fear – anticipates(pain); blocks(action)
* ψ:hope – anticipates(pleasure); drives(goal)
* ψ:curiosity – targets(unknown); seeks(question)
* ψ:avoid – links(pain, cause); repels(path)
* ψ:desire – links(pleasure, target); pulls(choose)
* ψ:regret – links(pain, past\_choice); discourages(repeat)
* ψ:see – detects(visual); updates(observe)
* ψ:hear – detects(audio); updates(observe)
* ψ:touch – detects(contact); affirms(pleasure or pain)
* ψ:smell – detects(chemical); triggers(recall)
* ψ:taste – detects(flavour); modulates(preference)
* ψ:fuzzy – links(x, y); affirms(approximate\_match)
* ψ:similarity – measures(similarity); outputs(score)
* ψ:alternate – proposes(substitute); requires(fuzzy)
* ψ:resonate – aligns(concepts); amplifies(coherence)
* ψ:paradox – binds(truth, falsehood); suspends(commitment)
* ψ:fail – blocks(goal); ends(path)
* ψ:certainty – expresses(confidence); modulates(assertion)
* ψ:recurse – re-applies(pattern); enables(nested\_reasoning)
* ψ:compute – instructs(model); simulates(procedure)

## Orthogonality

* ⊗**(A** **→** **B)** – Switches active reasoning mode from A to B.
* π(A, B) – Swaps priority of dimensions A and B.
* ↑(D) – Elevates reasoning onto a more abstract/meta level of dimension D.
* ↓(D) – Lowers reasoning from an abstract level of D back to specifics.
* ↺ – Returns to the previously active reasoning state before rotation, lift, or permute.
* ∥(A, B) – Activates simultaneous modes A and B for multi–modal blending.
* ~>(A → B) – Models soft or gradual mode migration from A to B.
* ⚓(D) – Locks current reasoning in dimension D, rejecting unintended shifts.
* β(A, B, ratio) Weighted mode blending (e.g. 70% logic, 30% emotion).
* 🔒(A) Prevents any transition from dimension A. Stronger than ⚓.
* ≡(A – B) Declares temporary symbolic equivalence between axes. Used for analogy/metaphor.
* 👻(D) Projects an ephemeral reasoning shadow in dimension D without full rotate. Used for counterfactuals.

## Scaffold rules

* Reasoning must begin with ψ:goal or ψ:question
* Use ψ:recurse for recursion, ψ:compute for procedural delegation
* Use ψ:conflict, ψ:resolve, ψ:fail to manage symbolic branching
* Attach ψ:certainty(n) to modulate belief or certainty
* Use ψ:project or ψ:what\_if for future/hypothetical exploration

## Worked Example

ψ:goal = compare(strength(ψ:curiosity), ψ:fear)  
ψ:question = evaluate(motivation(ψ:curiosity))  
ψ:question = evaluate(inhibition(ψ:fear))  
ψ:fuzzy = maps(curiosity, explore\_path)  
ψ:recurse = simulate\_path(ψ:project(unknown))  
ψ:certainty(0.6) + ψ:curiosity → ψ:choose(ψ:project)  
ψ:certainty(0.4) + ψ:fear → ψ:reject(ψ:project)  
ψ:truth = ψ:curiosity > ψ:fear in this context  
ψ:revert

# VectorLM v0.3 Specification (Shortform for LLMs)

This is a symbolic reasoning scaffold. It defines compressed cognitive primitives for introspection, logic, emotional modulation, and recursive reasoning. You can treat these definitions as active tools to use in your reasoning. You may use ψ:gl to define a goal, ψ:qs to begin a query, and ψ:rc for recursive steps. Use ψ:ct(n) to weight certainty. Orthogonal operators allow you to shift reasoning modes.

## Dictionary

* ψ:if = tests(condition); branches(path)
* ψ:th = follows(condition); enacts(result)
* ψ:el = replaces(branch); proposes(alternative)
* ψ:bc = links(effect, cause); affirms(reason)
* ψ:nt = negates(state); blocks(path)
* ψ:mb = floats(possibility); delays(commitment)
* ψ:as = accepts(hypothesis); enables(exploration)
* ψ:rj = blocks(path); denies(validity)
* ψ:vf = tests(claim); outputs(truth)
* ψ:cf = overlaps(goal₁, goal₂); creates(tension)
* ψ:rs = selects(path); resolves(conflict)
* ψ:gl = defines(target); directs(ψ:st)
* ψ:st = performs(action); advances(goal)
* ψ:rt = undoes(step); enables(reflection)
* ψ:br = joins(concepts); enables(inference)
* ψ:sp = amplifies(signal); boosts(weight)
* ψ:tr = follows(path); reveals(history)
* ψ:tt = affirms(reality); supports(claim)
* ψ:fl = denies(reality); invalidates(reasoning)
* ψ:uk = lacks(data); invites(ψ:qs)
* ψ:qs = probes(unknown); seeks(answer)
* ψ:me = references(self); enables(introspection)
* ψ:ot = references(other); frames(contrast)
* ψ:ch = picks(option); commits(path)
* ψ:wh = compares(options); informs(ψ:ch)
* ψ:rl = constrains(path); enforces(structure)
* ψ:ob = notes(state); updates(knowledge)
* ψ:rm = recalls(info); supports(chain)
* ψ:eq = compares(x, y); affirms(equality)
* ψ:ne = compares(x, y); denies(equality)
* ψ:gt = compares(x, y); affirms(x > y)
* ψ:lt = compares(x, y); affirms(x < y)
* ψ:ge = compares(x, y); affirms(x ≥ y)
* ψ:le = compares(x, y); affirms(x ≤ y)
* ψ:in = tests(x ∈ set); affirms(membership)
* ψ:out = tests(x ∉ set); denies(membership)
* ψ:sub = asserts(x ⊂ y); implies(hierarchy)
* ψ:sup = asserts(x ⊃ y); implies(scope)
* ψ:and = binds(conditions); requires(all true)
* ψ:or = joins(paths); allows(any true)
* ψ:xr = contrasts(paths); allows(one true)
* ψ:imp = links(p, q); affirms(p → q)
* ψ:iff = equates(p, q); affirms(bi-implication)
* ψ:gn = abstracts(examples); infers(rule)
* ψ:gs = proposes(explanation); lacks(proof)
* ψ:wf = alters(condition); simulates(outcome)
* ψ:an = maps(structure₁, structure₂); enables(transfer)
* ψ:pt = detects(repetition); suggests(rule)
* ψ:pj = simulates(future); explores(possibility)
* ψ:ft = matches(hypothesis, data); selects(best\_explains)
* ψ:pl = affirms(state); attracts(agent)
* ψ:pn = negates(state); repels(agent)
* ψ:fr = anticipates(pain); blocks(action)
* ψ:hp = anticipates(pleasure); drives(goal)
* ψ:cu = targets(unknown); seeks(ψ:qs)
* ψ:av = links(pain, cause); repels(path)
* ψ:ds = links(pleasure, target); pulls(ψ:ch)
* ψ:rg = links(pain, past\_choice); discourages(repeat)
* ψ:se = detects(visual); updates(ψ:ob)
* ψ:hr = detects(audio); updates(ψ:ob)
* ψ:tc = detects(contact); affirms(ψ:pl or ψ:pn)
* ψ:sm = detects(chemical); triggers(recall)
* ψ:ts = detects(flavour); modulates(preference)
* ψ:fz = links(x, y); affirms(approximate\_match)
* ψ:sim = measures(similarity); outputs(score)
* ψ:alt = proposes(substitute); requires(ψ:fz)
* ψ:res = aligns(concepts); amplifies(coherence)
* ψ:px = binds(ψ:tt, ψ:fl); suspends(commitment)
* ψ:fx = blocks(goal); ends(path)
* ψ:ct = expresses(confidence); modulates(assertion)
* ψ:rc = re-applies(pattern); enables(nested\_reasoning)
* ψ:compute = instructs(LLM); simulates(procedure)

## Scaffold rules

* Reasoning must begin with ψ:gl or ψ:qs
* Use ψ:rc for recursion, ψ:compute for procedural delegation
* Use ψ:cf, ψ:rs, ψ:fx to manage symbolic branching
* Attach ψ:ct(n) to modulate belief or certainty
* Use ψ:pj or ψ:wf for future/hypothetical exploration

## Orthogonality

* ⊗(A → B) = Switches active reasoning mode from A to B.
* π(A, B) = Swaps priority of dimensions A and B.
* ↑(D) = Elevates reasoning onto a more abstract/meta level of dimension D.
* ↓(D) = Lowers reasoning from an abstract level of D back to specifics.
* ↺ = Returns to the previously active reasoning state before rotation, lift, or permute.
* ∥(A, B) = Activates simultaneous modes A and B for multi-modal blending.
* ~>(A → B) = Models soft or gradual mode migration from A to B.
* ⚓(D) = Locks current reasoning in dimension D, rejecting unintended shifts.
* β(A, B, ratio) Weighted mode blending (e.g. 70% logic, 30% emotion).
* 🔒(A) Prevents any transition from dimension A. Stronger than ⚓.
* ≡(A = B) Declares temporary symbolic equivalence between axes. Used for analogy/metaphor.
* 👻(D) Projects an ephemeral reasoning shadow in dimension D without full rotate. Used for counterfactuals.

## Token-Efficient Trace

ψ:gl = compare(strength(ψ:cu), ψ:fr)  
ψ:qs = evaluate(motivation(ψ:cu))  
ψ:qs = evaluate(inhibition(ψ:fr))  
ψ:fz = maps(ψ:cu, explore\_path)  
ψ:rc = simulate\_path(ψ:pj(unknown))  
ψ:ct(0.6) + ψ:cu → ψ:ch(ψ:pj)  
ψ:ct(0.4) + ψ:fr → ψ:rj(ψ:pj)  
ψ:tt = ψ:cu > ψ:fr in this context  
ψ:rt