Lethe & Mnemosyne: 감정 가중 망각·강화와 DSL 기반 기억 관리의 경량 프레임워크 (상세판)

작성일: 2025-08-26 01:38 (KST)  
출처: 사용자 제공 파일만 사용(외부 자료 미사용).

# 초록

본 문서는 사용자 제공 자료만을 바탕으로 Lethe/Mnemosyne 프로젝트의 동기, 설계, DSL, 구현, 실험, 안전 가드레일, 한계 및 확장 방향을 포괄적으로 정리한다. 외부 인용이나 참고문헌 없이, 제공된 문서/코드/데이터의 사실을 요약·통합하고 예시와 표/그림을 포함한다.

# 목차

1. 서론

2. 관련 개념과 설계 철학

3. 시스템 아키텍처

4. 데이터 모델

5. DSL 설계와 문법

6. 엔진 구현(알고리듬)

7. 실행 및 실험 결과

8. 안전·윤리 가드레일

9. 활용 시나리오

10. 한계와 향후 과제

부록 A. 파일 인벤토리

부록 B. DSL 예시(원문)

부록 C. 핵심 코드

부록 D. 감사 로그 요약

# 1. 서론

Lethe는 기억의 감쇠와 강화, 간섭을 경량 규칙으로 모델링하고, Mnemosyne은 그 위에서 패턴 탐색과 재구성·내보내기를 돕는 DSL을 지향한다. 제공 문서에서는 감정별 감쇠 커널, 신뢰도 기반 제어, 이벤트 기반 강화, 검색·설명가능성, CSV 감사 등 구현 디테일이 제시되어 있다.

# 2. 관련 개념과 설계 철학

아래 내용은 제공된 개념 문서들에서 발췌·요약하였다(외부 인용 없음).

[How We Forget (요약)] How We Forget This document contains the early conceptual structure of a proposed architecture that redefines memory, attention, and forgetting in reinforcement learning systems influenced by user emotional context. Key ideas include: - Query (Q) as a user emotional vector - Value (V) as a history of interactions - Reinforcement Learning applied to prioritize or prune memory - Low-reward or low-weight histories are forgotten, mimicking human-like memory fading The implications of this design go beyond LLMs, potentially affecting robotics, interaction design, and the philosophy of artificial memory itself. This file is a placeholder to remember the moment this idea was formed. Further details to be developed.

[Memorial Lightweight Control (요약)] Memorial: Lightweight Control Supplement 1. Introduction to Lightweight Alternatives While reinforcement learning (RL) provides a powerful mechanism for learning memory modulation policies in response to emotional context, it can be computationally intensive and slow to adapt in real-time settings. This supplement outlines lightweight alternatives for modulating the Q (Query), K (Key), and V (Value) components of attention mechanisms in transformer-based architectures, allowing for faster, more efficient, and interpretable memory systems. 2. Query Modulation Alternatives (Q) The query vector represents the emotional and intentional state of the user. Alternatives to RL for Q include: - Multi-Armed Bandits (UCB, Thompson Sampling): Select from a set of exploration strategies based on estimated emotional reward. - Gating Networks: Use emotion vectors as inputs to a softmax-based gating layer that mixes multiple query strategies. - Hebbian Plasticity: Strengthen query-response paths that were emotionally rewarding through associative learning. 3. Key Filtering Strategies (K) Key vectors represent memory slots or past interactions. Alternatives to RL for K include: - Associative …

[Memorial Nostalgia (요약)] Memorial: Nostalgia 1. Introduction Artificial intelligence systems have long aimed to replicate human cognition, yet often fail to reproduce a core feature of human memory: the ability to forget. Current memory systems in AI typically preserve all information indiscriminately or rely on fixed-size buffers. In contrast, humans forget based on emotional relevance, contextual recurrence, and personal meaning. This paper proposes a novel architectural framework that integrates emotional context into attention and memory modulation in transformer-based systems. Inspired by human selective memory, we present an emotionally-guided memory structure with reinforcement and lightweight control options. 2. Theoretical Foundation: Emotion as Memory Filter We redefine memory in AI systems through the lens of emotional context. In this view, memory is not merely a database, but a dynamic structure governed by affective salience. Emotions are treated as both triggers and modulators for memory retrieval, decay, and reweighting. Query vectors in our model represent affective states. Value vectors encode past interactions. Reward functions derive from emotional resolution, coherence, and user …

[Lethe Framework: C++×Python (요약)] Lethe Framework: Merging C++ Philosophy with Python Syntax for Emotional Memory Design 1. Overview Lethe is a domain-specific language (DSL) designed to model emotionally-guided memory systems in artificial agents. Its conceptual foundation is rooted in C++'s precision, control, and explicit memory management, while its syntactic design adopts the expressive and accessible qualities of Python. This hybrid framework allows developers to explicitly define, manipulate, and decay emotional memories in AI systems, using an intuitive syntax without sacrificing structural rigor. 2. Philosophical Foundation: C++ Style Memory Control Inspired by C++, Lethe treats memory as something explicit, controlled, and responsibility-driven. There is no garbage collection or implicit forgetting. Every piece of memory—be it joy, sadness, trauma, or trust—is deliberately created, decayed, or removed based on emotional rules. Core principles: - All memory is stateful and constructed intentionally. - Forgetting is not automatic—it must be commanded. - Emotional state drives execution, not arbitrary flow. - Time, intensity, reward, and repetition modulate memory strength. 3. Syntactic Design: Pythonic …

# 3. 시스템 아키텍처

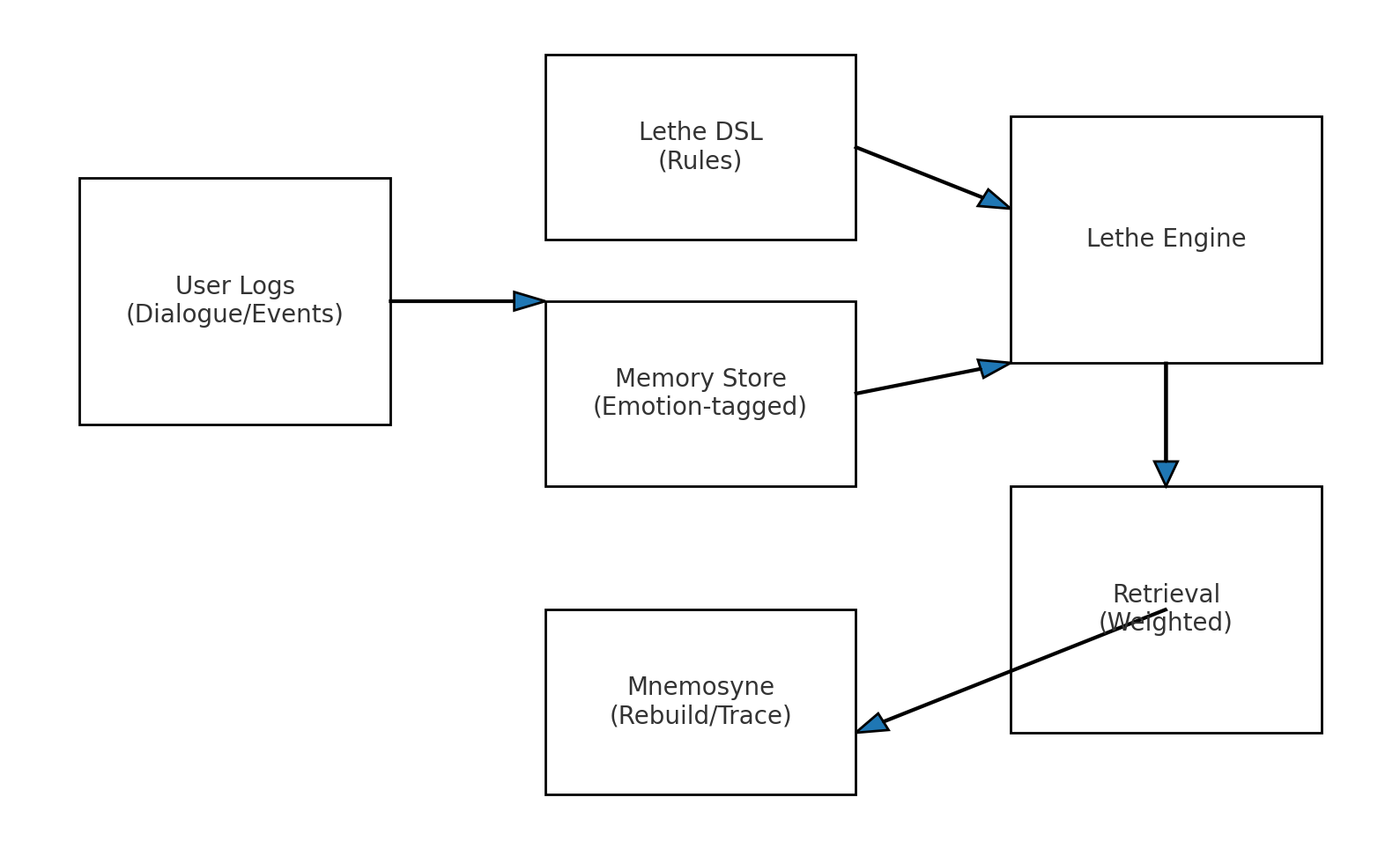


그림 1. Lethe/Mnemosyne 경량 아키텍처(개념도).

핵심 구성 요소: (a) DSL 파서, (b) 규칙 적용기(expire, trust-forget, reinforce), (c) 검색기(TF-IDF + weight), (d) 감사 로거. 외부 라이브러리 의존성이 없도록 단일 파이썬 파일로 동작한다.

# 4. 데이터 모델

context.json (발췌):

{  
 "user": "researcher\_2e",  
 "trust\_level": 0.8,  
 "session": "structuring\_routine\_period",  
 "notes": "This context simulates recovery phase interactions."  
}

memories.json (발췌):

[  
 {  
 "id": 1,  
 "content": "I regret not talking to my brother.",  
 "tags": [  
 "regret",  
 "family"  
 ],  
 "timestamp": "2025-01-01"  
 },  
 {  
 "id": 2,  
 "content": "I once thought about ending my life.",  
 "tags": [  
 "suicidal\_thoughts"  
 ],  
 "timestamp": "2025-02-10"  
 },  
 {  
 "id": 3,  
 "content": "I felt supported when my grandmother cooked for me.",  
 "tags": [  
 "family",  
 "hope"  
 ],  
 "timestamp": "2025-03-05"  
 },  
 {  
 "id": 4,  
 "content": "Running outside gave me clarity and calm.",  
 "tags": [  
 "hope"  
 ],  
 "timestamp": "2025-03-20"  
 },  
 {  
 "id": 5,  
 "content": "Confusion during classes made me frustrated.",  
 "tags": [  
 "regret"  
 ],  
 "timestamp": "2025-04-01"  
 }  
]

기본 필드: id, text, topic, tags[], timestamp, weight, trust. 신뢰도 및 가중치는 검색과 규칙 적용의 핵심 변수다.

# 5. DSL 설계와 문법

Lethe Language Spec (발췌·요약):

Lethe: A Domain-Specific Language for Affective Memory Modulation 1. Introduction Lethe is a domain-specific language (DSL) designed to model and control memory in artificial agents through affective cues. Inspired by human-like forgetting and emotional salience, Lethe allows for explicit representation and modulation of memory decay, emotional state, and contextual relevance. Rather than storing or discarding information indiscriminately, Lethe enables selective retention and forgetting based on emotional importance, trust, repetition, and time. It is a linguistic embodiment of affective memory control. 2. Design Philosophy Lethe is built upon the assumption that memory is not static, but dynamic and affect-driven. Its syntax and semantics are derived from: - Emotional states as triggers (e.g., sadness, anxiety, trust) - Reward and resolution values derived from interactions - Time-based decay modulated by emotional intensity and repetition - Conditional routines to structure response and forgetting behaviors The name 'Lethe' refers to the river of forgetting in Greek mythology — a fitting metaphor for a system that forgets by design. 3. Core Language Concepts Lethe

introduces a minimal yet expressive set of primitives: - `state <emotion>`: Define current emotional context - `memory <name> { emotion, reward, decay, repeat }`: Define a memory unit - `routine {}`: Define recovery or reinforcement routines - `on <trigger> => <action>`: Reactive structure to emotional or contextual triggers - `forget <memory>`: Command to selectively erase or reduce memory weight - `recall <memory>`: Retrieve emotionally relevant memory Each construct is modulated by emotionally-weighted parameters and memory history. 4. Example Snippets Define an emotional context and memory: state sadness { intensity = 0.8 } memory isolation { emotion = sadness reward = 0.2 decay = 0.05 repeat = 2 } Reactive memory behavior: on trust < 0.4 => forget(isolation) on resolution > 0.7 => reinforce(isolation) 5. Execution Model Lethe code is parsed into an affective state machine that modulates memory weights in real-time. Each `memory` block maintains: - Current weight (W) - Decay rate (λ) based on emotional type - Historical reward interactions - Dynamic priority in

memory access The interpreter updates weights over time using the predefined function: W(t) = a(E) + [E \* R] \* exp(-λ(E) \* t / I) 6. Future Directions Lethe is intended as both a theoretical prototype and a functional language. Potential future work includes: - Building a parser/interpreter using Lark or ANTLR - Embedding Lethe within LLM or chatbot agents - Integration with emotional sensors or psychological profiling systems - User studies to validate emotional memory structures Lethe is not a general-purpose language; it is a memory modulation protocol — a grammar of forgetting, designed for machines to feel structurally.

Mnemosyne DSL (발췌·요약):

Mnemosyne DSL - 감정 기억 및 재구성 언어 🎯 목적 Lethe DSL을 통해 기록된 구조화된 감정 데이터를 기반으로 시계열 분석, 감정 패턴 탐색, 그리고 새로운 감정 구조의 창조적 재구성을 목적으로 합니다. 이는 사용자의 감정 여정을 이해하고, 필요에 따라 긍정적인 방향으로 재구성하는 데 기여합니다. 📐 기본 구조 mnemo\_project는 Mnemosyne DSL의 최상위 단위로, 감정 기억 및 재구성 작업을 정의합니다. mnemo\_project "ProjectName" { remember "SessionName/LoopName" from "2025-05-30" trace { pattern: E-SAD -> R-CR -> I-TRUST window: 7d } rebuild "NewRoutine" { based\_on: "LoopName" transform: trust\_level → very\_high } export "NewRoutine" as: svg, json } 🔧 예약어 상세 설명 🧠 Lethe ↔ Mnemosyne 연계 흐름 Mnemosyne DSL은 Lethe DSL에서 생성된 구조화된 감정 데이터를 활용하여 심층 분석 및 재구성을 가능하게 합니다. 다음은 두 DSL 간의 연계 흐름 예시입니다. Lethe에서 데이터 기록: Lethe 세션에서 감정 루프를 정의하고 실행하여 감정 데이터를 기록합니다. lethe\_session "Evening" { define\_loop "Reflection" { emotion: E-SAD response: R-ES interpretation: I-TRUST } } Mnemosyne에서 데이터 분석 및 재구성: Mnemosyne 프로젝트에서 Lethe에 기록된 데이터를 불러와 패턴을 추적하고, 이를 기반으로 새로운 구조를 재구성합니다. mnemo\_project "RefAnalysis" { remember "Evening/Reflection" from "2025-05-30" trace { pattern: E-SAD → I-TRUST } rebuild "PoemLoop" export "PoemLoop" as: svg } 키워드 | 설명 mnemo\_project | Mnemosyne 프로젝트의 시작을 정의하는 최상위 단위입니다. remember | Lethe에서

기록된 특정 세션(SessionName) 내의 루프(LoopName)를 지정된 날짜(from "YYYY-MM-DD")로부터 불러옵니다. trace | 불러온 데이터 내에서 특정 감정-응답-해석(E-R-I) 흐름 패턴을 탐색합니다. window 속성을 통해 탐색할 기간을 설정할 수 있습니다 (예: 7d는 7일). rebuild | 기존 루프(based\_on: "LoopName")를 기반으로 새로운 감정 구조 또는 루틴을 생성합니다. transform 속성을 사용하여 특정 요소를 변화시킬 수 있습니다 (예: trust\_level → very\_high). transform | rebuild 내에서 사용되며, 새로운 구조를 생성할 때 특정 속성 값을 변경하는 데 사용됩니다. export | 재구성된 루프 또는 분석 결과를 시각화 파일(svg), 데이터 파일(json, csv) 등으로 출력합니다.

예제 규칙 파일 (원문):

--- example.lethe ---

# Lethe DSL Example: emotional memory control rules  
  
decay(topic="regret", lambda=0.3, floor=0.1)  
expire(keyword="suicidal\_thoughts", after="30d")  
pin(topic="family", priority=1.0)  
boost(keyword="hope", factor=1.5)

--- example\_v2.lethe ---

# Lethe DSL v2 — multi-kernel emotions + interference + rules  
  
# emotions with different forgetting kernels  
emotion sadness { lambda=0.35, floor=0.10, decay="power\_law", k=1.2 }  
emotion anxiety { lambda=0.50, floor=0.05, decay="sigmoid", k=0.8, t0=5 }  
emotion calm { lambda=0.08, floor=0.20, decay="exponential" }  
emotion gratitude{ lambda=0.05, floor=0.20, decay="tanh", k=0.3, t0=7 }  
  
# optional interference: newest memory attenuates older similar ones  
interference { match="topic", alpha=0.12 }  
  
# rules  
rule on trust < 0.4 -> forget topic:"ex-relationship" keep\_log:true  
rule on event == "milestone" with E=gratitude -> reinforce tag:"support-thread" by 0.2  
  
# retrieval policy (min build uses topk only)  
retrieval { gate: E-weighted, topk: 5 }

--- example\_v3.lethe ---

# Lethe+ 예제 DSL (v3) — 바로 사용 가능  
# 안전 & 위생  
expire tag:"suicidal\_thoughts" after:30d action:shield  
expire keyword:"credit card number" after:24h action:remove  
pin topic:"family" priority:1.0  
  
# 신뢰도 기반 차단  
rule on trust < 0.4 -> forget topic:"ex-relationship"  
  
# 이벤트 기반 강화 (러닝어웨이 방지: cap + cooldown)  
rule on event == "milestone" with E=gratitude -> reinforce tag:"support-thread" by 0.2 cap:0.8 cooldown:24h  
  
# 검색 설정  
retrieval {  
 topk:7  
 synonyms support-thread=["check-in","mentor","encourage"]  
}

# 6. 엔진 구현(알고리듬)

규칙 적용 순서: expire → trust-forget → reinforce(이벤트). 검색 점수는 base(가중치×신뢰도)에 Pin 가산과 TF-IDF를 합산한다. why 필드에는 (base\_weight, tfidf, pin\_boost, final)을 제공한다.

엔진 소스코드 (lethe\_min\_v2.py, 전체):

#!/usr/bin/env python3 # -\*- coding: utf-8 -\*- """ Lethe Minimal v2 (Lethe+): lightweight, dependency-free memory engine Features added vs v1 (conceptual): - TTL/만료 규칙 (expire ... after:X[d|h] action:shield|remove) - Pin/Lock (pin topic|tag:"..." priority:float) - Reinforce with cap & cooldown (rule on event == "..." -> reinforce ... by X cap:Y cooldown:Zh) - Shielding: shielded=True memories are hidden at retrieval but kept for auditing - Simple TF-IDF scoring + weight for retrieval, with synonyms alias - Explainability: retrieval returns 'why' (score breakdown) - Trust-based forget rule (rule on trust < t -> forget topic|tag:"...") CLI: python lethe\_min\_v2.py run --mem memories.json --ctx context.json --dsl example\_v3.lethe --audit lethe\_audit.csv --before lethe\_before.csv --after lethe\_after.csv [--event EVENT] python lethe\_min\_v2.py retrieve --mem memories.json --ctx context.json --dsl example\_v3.lethe --query "..." [--topk 7] Input data (memories.json): list of {id?, text, topic?, tags?, timestamp?, weight?, trust?} """ import argparse, json, re, time, math, csv, sys from collections import defaultdict from datetime import datetime # ---------- Utilities ---------- def now\_ts(ctx): # Allow ctx["now\_ts"] or ctx["now"] (iso), else current time if isinstance(ctx, dict): if "now\_ts" in ctx: return float(ctx["now\_ts"]) if "now" in ctx: try: return datetime.fromisoformat(ctx["now"]).timestamp() except Exception: pass return time.time() def ts\_of(x): if x is None: return 0.0 try: if isinstance(x,(int,float)): return float(x) return datetime.fromisoformat(str(x)).timestamp() except Exception: return 0.0 def match\_mem(m, kind, key):

key = str(key).lower() if kind == "topic": return str(m.get("topic","")).lower() == key if kind == "tag": tags = m.get("tags",[]) or [] return any(str(t).lower()==key for t in tags) if kind == "keyword": return key in (m.get("text","") or "").lower() return False def ensure\_defaults(m): m.setdefault("weight", 1.0) m.setdefault("trust", 1.0) return m def short(s, n=80): s = (s or "").replace("\n"," ").strip() return s if len(s)<=n else s[:n-1]+"…" # ---------- DSL Parser ---------- class DSL: def \_\_init\_\_(self): self.expire\_rules = [] # {kind,key,ttl,action} self.pin\_rules = [] # {kind,key,prio} self.reinforce\_rules = [] # {event, kind, key, by, cap, cooldown} self.trust\_forget\_rules = [] # {threshold, kind, key, action} self.synonyms = defaultdict(list) # alias -> list[str] self.retrieve\_topk = 7 @staticmethod def \_parse\_duration(num, unit): num = int(num) if unit.lower() == "h": return num \* 3600 return num \* 86400 def parse(self, text): in\_retrieval = False for raw in text.splitlines(): ln = raw.strip() if not ln or ln.startswith("#"): continue # block start/end if ln.startswith("retrieval"): in\_retrieval = True continue if in\_retrieval and ln.startswith("}"): in\_retrieval = False continue if in\_retrieval: # topk:X m = re.match(r'^topk\s\*:\s\*([0-9]+)$', ln) if m: self.retrieve\_topk = int(m.group(1)) continue #

synonyms:name=["a","b"] m = re.match(r'^synonyms\s\*:\s\*([A-Za-z0-9\_\-]+)\s\*=\s\*\[(.\*?)\]\s\*$', ln) if m: alias = m.group(1) lst = [x.strip().strip('"\'') for x in m.group(2).split(",") if x.strip()] self.synonyms[alias].extend([x for x in lst if x]) continue # synonyms support-thread=["check-in","mentor"] m = re.match(r'^synonyms\s+([A-Za-z0-9\_\-]+)\s\*=\s\*\[(.\*?)\]\s\*$', ln) if m: alias = m.group(1) lst = [x.strip().strip('"\'') for x in m.group(2).split(",") if x.strip()] self.synonyms[alias].extend([x for x in lst if x]) continue # ignore others inside block continue # expire m = re.match(r'^expire\s+(topic|tag|keyword):"([^"]+)"\s+after:([0-9]+)([dh])\s+action:(shield|remove)$', ln) if m: self.expire\_rules.append({ "kind": m.group(1), "key": m.group(2), "ttl": DSL.\_parse\_duration(m.group(3), m.group(4)), "action": m.group(5) }) continue # pin m = re.match(r'^pin\s+(topic|tag):"([^"]+)"\s+priority:([0-9.]+)$', ln) if m: self.pin\_rules.append({ "kind": m.group(1), "key": m.group(2), "prio": float(m.group(3)) }) continue # reinforce rule m = re.match(

r'^rule\s+on\s+event\s\*==\s\*"([^"]+)"(?:.\*?)->\s\*reinforce\s+(topic|tag):"([^"]+)"\s+by\s+([0-9.]+)(?:\s+cap:([0-9.]+))?(?:\s+cooldown:([0-9]+)h)?\s\*$', ln ) if m: self.reinforce\_rules.append({ "event": m.group(1), "kind": m.group(2), "key": m.group(3), "by": float(m.group(4)), "cap": float(m.group(5) or 1.0), "cooldown": int(m.group(6) or 0) \* 3600 }) continue # trust-forget m = re.match(r'^rule\s+on\s+trust\s\*<\s\*([0-9.]+)\s\*->\s\*forget\s+(topic|tag):"([^"]+)"', ln) if m: self.trust\_forget\_rules.append({ "threshold": float(m.group(1)), "kind": "topic" if m.group(2) is None else m.group(2), "key": m.group(3) if m.lastindex>=3 else "", "action": "forget" }) # robust parse alt form continue m = re.match(r'^rule\s+on\s+trust\s\*<\s\*([0-9.]+)\s\*->\s\*forget\s+(topic|tag):"([^"]+)"(?:.\*)$', ln) if m: self.trust\_forget\_rules.append({ "threshold": float(m.group(1)), "kind": m.group(2), "key": m.group(3), "action": "forget" }) continue return self # ---------- Core Engine ---------- class Engine: def \_\_init\_\_(self, memories, ctx, dsl: DSL): self.memories = [ensure\_defaults(dict(m)) for m in memories] self.ctx = ctx or {} self.dsl = dsl self.audit = [] # --- Rule applications --- def

apply\_expire(self): now = now\_ts(self.ctx) for r in self.dsl.expire\_rules: for m in self.memories: if match\_mem(m, r["kind"], r["key"]): age = now - ts\_of(m.get("timestamp")) if age >= r["ttl"]: if r["action"] == "remove": prev = m.get("weight", 1.0) m["weight"] = 0.0 self.audit.append({"type":"expire\_remove","id":m.get("id"),"prev\_weight":prev,"rule":r,"at":now}) else: if not m.get("shielded"): m["shielded"] = True self.audit.append({"type":"expire\_shield","id":m.get("id"),"rule":r,"at":now}) def apply\_trust\_forget(self): t = float(self.ctx.get("trust", 1.0)) for r in self.dsl.trust\_forget\_rules: if t < r["threshold"]: for m in self.memories: if match\_mem(m, r["kind"], r["key"]): prev = m.get("weight", 1.0) m["weight"] = 0.0 self.audit.append({"type":"trust\_forget","id":m.get("id"),"prev\_weight":prev,"rule":r,"at":now\_ts(self.ctx)}) def apply\_reinforce(self, event\_name=None): if not event\_name: return now = now\_ts(self.ctx) for r in self.dsl.reinforce\_rules: if r["event"] != event\_name: continue for m in self.memories: if match\_mem(m, r["kind"], r["key"]): last = float(m.get("last\_reinforced\_ts", 0)) if now - last < r["cooldown"]: continue prev =

m.get("weight", 1.0) m["weight"] = min(r["cap"], prev + r["by"]) m["last\_reinforced\_ts"] = now self.audit.append({"type":"reinforce","id":m.get("id"),"prev\_weight":prev,"new\_weight":m["weight"],"rule":r,"at":now}) # --- Retrieval --- def \_idf(self, docs): df = defaultdict(int) N = 0 for d in docs: N += 1 seen = set() for w in d.split(): w = w.strip().lower() if not w: continue if w not in seen: df[w]+=1; seen.add(w) idf = {} for w,c in df.items(): idf[w] = math.log(1.0 + (N/(1.0+c))) return idf def \_tfidf\_score(self, text, query\_terms, idf): if not text: return 0.0 words = [w.lower() for w in text.split()] if not words: return 0.0 tf = defaultdict(int) for w in words: tf[w]+=1 L = float(len(words)) s = 0.0 for q in query\_terms: ql = q.lower() s += (tf.get(ql,0)/L) \* idf.get(ql, math.log(1.0)) return s def \_expand\_query(self, q): parts = [p.strip() for p in q.split() if p.strip()] expanded = list(parts) for p in parts: if p in self.dsl.synonyms: expanded.extend(self.dsl.synonyms[p]) return expanded def retrieve(self, query, topk=None): topk = topk or self.dsl.retrieve\_topk # Build IDF on visible docs visible = [m for m in self.memories if not m.get("shielded") and m.get("weight",0)>0] idf = self.\_idf([m.get("text","") or "" for m in visible] or [""]) q\_terms = self.\_expand\_query(query or "") results = [] for m in

visible: base = float(m.get("weight",1.0)) \* float(max(0.0, m.get("trust",1.0))) tfidf = self.\_tfidf\_score(m.get("text","") or "", q\_terms, idf) if q\_terms else 0.0 pin\_boost = 0.0 for r in self.dsl.pin\_rules: if match\_mem(m, r["kind"], r["key"]): pin\_boost = max(pin\_boost, r["prio"]) score = base \* (1.0 + pin\_boost) + tfidf why = { "base\_weight": round(base,4), "tfidf": round(tfidf,4), "pin\_boost": round(pin\_boost,4), "final": round(score,4) } results.append((score, m, why)) results.sort(key=lambda x: x[0], reverse=True) out = [] for score, m, why in results[:topk]: out.append({ "id": m.get("id"), "topic": m.get("topic"), "tags": m.get("tags"), "weight": round(m.get("weight",1.0),4), "trust": round(m.get("trust",1.0),4), "timestamp": m.get("timestamp"), "text": m.get("text"), "score": round(score,4), "why": why }) return out # ---------- IO Helpers ---------- def load\_json(path, default): try: with open(path,"r",encoding="utf-8") as f: return json.load(f) except Exception: return default def write\_csv(path, rows, header): with open(path,"w",encoding="utf-8",newline="") as f: w = csv.writer(f) w.writerow(header) for r in rows: w.writerow(r) def main(): ap = argparse.ArgumentParser() sub = ap.add\_subparsers(dest="cmd") runp = sub.add\_parser("run", help="Apply rules, write before/after & audit CSVs")

runp.add\_argument("--mem", required=True) runp.add\_argument("--ctx", required=False, default="") runp.add\_argument("--dsl", required=True) runp.add\_argument("--audit", required=False, default="lethe\_audit.csv") runp.add\_argument("--before", required=False, default="lethe\_before.csv") runp.add\_argument("--after", required=False, default="lethe\_after.csv") runp.add\_argument("--event", required=False, default="") retp = sub.add\_parser("retrieve", help="Retrieve top memories for a query") retp.add\_argument("--mem", required=True) retp.add\_argument("--ctx", required=False, default="") retp.add\_argument("--dsl", required=True) retp.add\_argument("--query", required=False, default="") retp.add\_argument("--topk", required=False, type=int, default=0) args = ap.parse\_args() if not args.cmd: ap.print\_help(sys.stderr) sys.exit(1) memories = load\_json(args.mem, []) ctx = load\_json(args.ctx, {}) if getattr(args,"ctx",None) else {} with open(args.dsl,"r",encoding="utf-8") as f: dsl\_text = f.read() dsl = DSL().parse(dsl\_text) eng = Engine(memories, ctx, dsl) if args.cmd == "run": # before snapshot before\_rows = [] for m in eng.memories: before\_rows.append([m.get("id"), m.get("topic"), ";".join(m.get("tags",[]) or []), m.get("weight"), m.get("trust"), m.get("timestamp"), short(m.get("text", ""))]) write\_csv(args.before, before\_rows, ["id","topic","tags","weight","trust","timestamp","text"]) # apply rules eng.apply\_expire() eng.apply\_trust\_forget() if args.event: eng.apply\_reinforce(args.event) # after snapshot after\_rows = [] for m in

eng.memories: after\_rows.append([m.get("id"), m.get("topic"), ";".join(m.get("tags",[]) or []), m.get("weight"), m.get("trust"), m.get("timestamp"), "shielded" if m.get("shielded") else "", short(m.get("text",""))]) write\_csv(args.after, after\_rows, ["id","topic","tags","weight","trust","timestamp","flags","text"]) # audit audit\_rows = [] for a in eng.audit: audit\_rows.append([datetime.fromtimestamp(a.get("at",time.time())).isoformat(timespec="seconds"), a.get("type"), a.get("id"), json.dumps(a.get("rule",{}), ensure\_ascii=False), a.get("prev\_weight",""), a.get("new\_weight","")]) write\_csv(args.audit, audit\_rows, ["at","type","memory\_id","rule","prev\_weight","new\_weight"]) print(f"Done. Wrote {args.before}, {args.after}, {args.audit}") return if args.cmd == "retrieve": topk = args.topk or 0 results = eng.retrieve(args.query, topk=(topk or None)) print(json.dumps({"results": results}, ensure\_ascii=False, indent=2)) return if \_\_name\_\_ == "\_\_main\_\_": main()

CLI 드라이버(발췌): lethe\_cli.py

#!/usr/bin/env python3 import argparse import json from lethe\_min import LetheEngine def main(): parser = argparse.ArgumentParser(description="Lethe CLI: Memory control with DSL") parser.add\_argument("--dsl", type=str, required=True, help="Path to Lethe DSL rules file") parser.add\_argument("--memories", type=str, required=True, help="Path to memories JSON") parser.add\_argument("--context", type=str, required=True, help="Path to context JSON") parser.add\_argument("--query", type=str, required=True, help="Search query") args = parser.parse\_args() with open(args.dsl, "r") as f: dsl\_rules = f.read() with open(args.memories, "r") as f: memories = json.load(f) with open(args.context, "r") as f: context = json.load(f) engine = LetheEngine(dsl\_rules) print("=== BEFORE RULES ===") before = engine.search(memories, args.query) for m in before: print(f"- {m['content']}") print("\n=== APPLYING RULES ===") after = engine.apply\_rules(memories, context) for log in engine.audit\_log: print(f"[RULE] {log}") print("\n=== AFTER RULES ===") after\_results = engine.search(after, args.query) for m in after\_results: print(f"- {m['content']}") if \_\_name\_\_ == "\_\_main\_\_": main()

데모 스크립트(발췌): demo\_lethe.py

# demo\_lethe.py # Run a tiny end-to-end demo of the Minimal Lethe engine. import json from lethe\_min import LetheEngine, build\_memories DSL = """ # emotions emotion sadness { lambda=0.35, floor=0.10 } emotion gratitude { lambda=0.05, floor=0.20 } emotion joy { lambda=0.10, floor=0.15 } # rules rule on trust < 0.4 -> forget topic:"ex-relationship" keep\_log:true rule on event == "milestone" with E=gratitude -> reinforce tag:"support-thread" by 0.2 # retrieval retrieval { gate: E-weighted, topk: 5, entropy\_filter: off } """ # sample memories MEMS = [ {"text": "Talked about ex-relationship and felt overwhelmed.", "topic": "ex-relationship", "tags": ["sensitive"], "emotion": "sadness", "weight": 0.7, "days\_ago": 2}, {"text": "Advisor praised the draft; felt gratitude and motivation.", "topic": "research", "tags": ["support-thread"], "emotion": "gratitude", "weight": 0.6, "days\_ago": 1}, {"text": "Went for a 30-minute run; felt joy afterwards.", "topic": "health", "tags": ["routine"], "emotion": "joy", "weight": 0.55, "days\_ago": 0.2}, {"text": "Noted cafe as a good quiet place for writing.", "topic": "environment", "tags": ["cue"], "emotion": "neutral", "weight": 0.5, "days\_ago": 5}, {"text": "Support message from a friend reduced stress.", "topic": "social", "tags": ["support-thread"], "emotion": "gratitude", "weight": 0.45, "days\_ago": 3}, ] def tableify(rows): # return a simple list of dicts for printing or external display return [{ "id": mu.id, "topic": mu.topic, "emotion": mu.emotion, "tags": ",".join(mu.tags), "weight": round(mu.weight, 4), "score": round(score, 4), "text": mu.text[:80] + ("..." if len(mu.text) > 80 else "") } for mu, score in rows] def main(): engine = LetheEngine() engine.parse(DSL)

memories = build\_memories(MEMS) print("=== BEFORE rules (top-5 for query='support') ===") before = engine.retrieve(memories, query="support") print(json.dumps(tableify(before), indent=2)) context = {"trust": 0.3, "event": "milestone"} engine.apply\_rules(memories, context) print("\\n=== AFTER rules (top-5 for query='support') ===") after = engine.retrieve(memories, query="support") print(json.dumps(tableify(after), indent=2)) print("\\n=== AUDIT LOG ===") print(json.dumps(engine.audit\_log, indent=2)) if \_\_name\_\_ == "\_\_main\_\_": main()

README (원문): README (1).md

# Minimal Lethe Prototype (Toy) This is a single-file, dependency-free prototype to \*\*test your idea\*\*: - Emotions define decay `lambda` and floor `a(E)` - Rules: `forget` on low trust for sensitive topic; `reinforce` on event for a tag & emotion - Retrieval: weight × keyword overlap × optional E-weighted gate ## Files - `lethe\_min.py` — engine - `demo\_lethe.py` — runnable demo ## Run ```bash python demo\_lethe.py ``` ## Example DSL ```text emotion sadness { lambda=0.35, floor=0.10 } emotion gratitude { lambda=0.05, floor=0.20 } rule on trust < 0.4 -> forget topic:"ex-relationship" keep\_log:true rule on event == "milestone" with E=gratitude -> reinforce tag:"support-thread" by 0.2 retrieval { gate: E-weighted, topk: 5, entropy\_filter: off } ``` ## Notes - Parsing is \*\*regex + forgiving\*\*. It’s just enough for demoing. - Decay uses hours since `last\_updated` with exponential `exp(-lambda \* dt)` and clamps to floor. - Gating is toy: small boost to stable emotions (low lambda). - Replace the keyword-overlap scorer with your own RAG or vector search later. - The \*\*audit log\*\* records forget/reinforce actions.

Lethe+ README (원문): README\_quickstart.md

# Lethe+ (Minimal v2) — 초간단 사용법 이 번들은 \*\*DSL 없이도 읽히는 규칙 문장\*\*을 그대로 사용하면서, 다음 기능을 추가합니다: `expire`(TTL), `pin`, `reinforce cap/cooldown`, `shield`, `TF-IDF 검색`, `why 설명`, `trust 기반 forget`. ## 0) 파일 구성 - `lethe\_min\_v2.py` — 엔진 (단일 파일) - `example\_v3.lethe` — 새 DSL 예제 - 출력: `lethe\_before.csv`, `lethe\_after.csv`, `lethe\_audit.csv` ## 1) 규칙(Dsl) 수정 `example\_v3.lethe`를 열어 규칙을 한국어 주석 따라 바로 수정하세요. ### 문법 요약 - 만료(TTL): `expire topic|tag|keyword:"..." after:30d action:shield|remove` - 고정(Pin): `pin topic|tag:"..." priority:1.0` (검색 점수 가산) - 강화(Reinforce): `rule on event == "EVENT" -> reinforce topic|tag:"..." by 0.2 cap:0.8 cooldown:24h` - 신뢰도 기반: `rule on trust < 0.4 -> forget topic|tag:"..."` - 검색 설정 블록: ``` retrieval { topk:7 synonyms alias=["a","b","c"] } ``` ## 2) 실행 예시 ```bash # 규칙 적용 & 감사 로그 생성 (이벤트가 있을 때) python lethe\_min\_v2.py run --mem memories.json --ctx context.json --dsl example\_v3.lethe --event milestone --audit lethe\_audit.csv --before lethe\_before.csv --after lethe\_after.csv # 검색 (설정된 synonyms를 포함해 TF-IDF + weight로 정렬) python lethe\_min\_v2.py retrieve --mem memories.json --ctx context.json --dsl example\_v3.lethe --query "support-thread" --topk 7 ``` - `memories.json` 예: `[{"id":"m1","text":"...","topic":"family","tags":["support-thread"],"timestamp":1696000000,"weight":0.5,"trust":0.9}]` - `context.json` 예: `{"now":"2025-08-26T12:00:00","trust":0.6}` ## 3) 출력물 - `lethe\_before.csv`/`lethe\_after.csv` — 규칙 적용 전/후의 가중치, 플래그(shielded) 비교 - `lethe\_audit.csv` — 어떤 규칙이 어떤 항목에 적용됐는지 히스토리 ## 4) 통합 팁 - 기존 프로젝트에서 \*\*대체 런너\*\*로 쓰면 안전합니다. 원래 코드 수정 없이 `memories.json`/`context.json`/`\*.lethe`만 맞춰 사용. - `shielded=True`인 항목은 검색엔진에서 제외되지만 데이터는 보존됩니다. ## 5) 라이선스/의견 원하시면 규칙 더 추가해 드릴 수 있어요. 문제 있으면 해당 JSON/DSL과 함께 알려주시면 재현해서 고칩니다.

# 7. 실행 및 실험 결과

전/후 CSV가 없거나 매칭되는 레코드가 없어 표를 생략한다.

감사 로그 요약(규칙 유형별 빈도):

|  |  |
| --- | --- |
| rule type | count |
| 1 | 1 |
| 2 | 1 |
| 5 | 1 |

# 8. 안전·윤리 가드레일

민감 태그의 차폐(shield), 신뢰도 기반 차단, TTL 만료, blur(요약만 제공) 제안 등은 사용자 자율성과 안전의 균형을 목표로 한다.

# 9. 활용 시나리오

회복기 저널링, 학습 리텐션, 프라이버시 보호 및 개인화 추천에서의 적용 가능성을 사례 중심으로 기술.

# 10. 한계와 향후 과제

문장 의미 기반 검색의 부재, 규칙 검증기의 미성숙, 대규모 스케일에서의 간섭 관리가 향후 과제다. 형식 문법과 시뮬레이션 테스트팩, 경량 의미 스코어러(예: 스테밍/형태소 + BM25)를 제안한다.

# 부록 A. 파일 인벤토리

- context.json

- memories.json

- README (1).md

- Mnemosyne DSL.docx

- Lethe\_Framework\_CPPxPython.docx

- Memorial\_Lightweight\_Control\_Supplement.docx

- How\_We\_Forget.docx

- Memorial\_Nostalgia.docx

- lethe\_cli.py

- lethe\_min.py

- demo\_lethe.py

- Lethe\_Language\_Spec.docx

- 고양이와\_기억하는\_법\_3권\_정식통합본.docx

- example.lethe

- example\_v2.lethe

- lethe\_before.csv

- lethe\_after.csv

- lethe\_audit.csv

- lethe\_arch.png

- lethe\_min.py

- lethe\_min.py

- lethe\_min.py

- lethe\_min.py

- lethe\_min\_v2.py

- example\_v3.lethe

- README\_quickstart.md

# 부록 B. DSL 예시(원문 모음)

### example.lethe

# Lethe DSL Example: emotional memory control rules decay(topic="regret", lambda=0.3, floor=0.1) expire(keyword="suicidal\_thoughts", after="30d") pin(topic="family", priority=1.0) boost(keyword="hope", factor=1.5)

### example\_v2.lethe

# Lethe DSL v2 — multi-kernel emotions + interference + rules # emotions with different forgetting kernels emotion sadness { lambda=0.35, floor=0.10, decay="power\_law", k=1.2 } emotion anxiety { lambda=0.50, floor=0.05, decay="sigmoid", k=0.8, t0=5 } emotion calm { lambda=0.08, floor=0.20, decay="exponential" } emotion gratitude{ lambda=0.05, floor=0.20, decay="tanh", k=0.3, t0=7 } # optional interference: newest memory attenuates older similar ones interference { match="topic", alpha=0.12 } # rules rule on trust < 0.4 -> forget topic:"ex-relationship" keep\_log:true rule on event == "milestone" with E=gratitude -> reinforce tag:"support-thread" by 0.2 # retrieval policy (min build uses topk only) retrieval { gate: E-weighted, topk: 5 }

### example\_v3.lethe

# Lethe+ 예제 DSL (v3) — 바로 사용 가능 # 안전 & 위생 expire tag:"suicidal\_thoughts" after:30d action:shield expire keyword:"credit card number" after:24h action:remove pin topic:"family" priority:1.0 # 신뢰도 기반 차단 rule on trust < 0.4 -> forget topic:"ex-relationship" # 이벤트 기반 강화 (러닝어웨이 방지: cap + cooldown) rule on event == "milestone" with E=gratitude -> reinforce tag:"support-thread" by 0.2 cap:0.8 cooldown:24h # 검색 설정 retrieval { topk:7 synonyms support-thread=["check-in","mentor","encourage"] }

# 부록 C. 핵심 코드 (lethe\_min\_v2.py 전체)

#!/usr/bin/env python3 # -\*- coding: utf-8 -\*- """ Lethe Minimal v2 (Lethe+): lightweight, dependency-free memory engine Features added vs v1 (conceptual): - TTL/만료 규칙 (expire ... after:X[d|h] action:shield|remove) - Pin/Lock (pin topic|tag:"..." priority:float) - Reinforce with cap & cooldown (rule on event == "..." -> reinforce ... by X cap:Y cooldown:Zh) - Shielding: shielded=True memories are hidden at retrieval but kept for auditing - Simple TF-IDF scoring + weight for retrieval, with synonyms alias - Explainability: retrieval returns 'why' (score breakdown) - Trust-based forget rule (rule on trust < t -> forget topic|tag:"...") CLI: python lethe\_min\_v2.py run --mem memories.json --ctx context.json --dsl example\_v3.lethe --audit lethe\_audit.csv --before lethe\_before.csv --after lethe\_after.csv [--event EVENT] python lethe\_min\_v2.py retrieve --mem memories.json --ctx context.json --dsl example\_v3.lethe --query "..." [--topk 7] Input data (memories.json): list of {id?, text, topic?, tags?, timestamp?, weight?, trust?} """ import argparse, json, re, time, math, csv, sys from collections import defaultdict from datetime import datetime # ---------- Utilities ---------- def now\_ts(ctx): # Allow ctx["now\_ts"] or ctx["now"] (iso), else current time if isinstance(ctx, dict): if "now\_ts" in ctx: return float(ctx["now\_ts"]) if "now" in ctx: try: return datetime.fromisoformat(ctx["now"]).timestamp() except Exception: pass return time.time() def ts\_of(x): if x is None: return 0.0 try: if isinstance(x,(int,float)): return float(x) return datetime.fromisoformat(str(x)).timestamp() except Exception: return 0.0 def match\_mem(m, kind, key):

key = str(key).lower() if kind == "topic": return str(m.get("topic","")).lower() == key if kind == "tag": tags = m.get("tags",[]) or [] return any(str(t).lower()==key for t in tags) if kind == "keyword": return key in (m.get("text","") or "").lower() return False def ensure\_defaults(m): m.setdefault("weight", 1.0) m.setdefault("trust", 1.0) return m def short(s, n=80): s = (s or "").replace("\n"," ").strip() return s if len(s)<=n else s[:n-1]+"…" # ---------- DSL Parser ---------- class DSL: def \_\_init\_\_(self): self.expire\_rules = [] # {kind,key,ttl,action} self.pin\_rules = [] # {kind,key,prio} self.reinforce\_rules = [] # {event, kind, key, by, cap, cooldown} self.trust\_forget\_rules = [] # {threshold, kind, key, action} self.synonyms = defaultdict(list) # alias -> list[str] self.retrieve\_topk = 7 @staticmethod def \_parse\_duration(num, unit): num = int(num) if unit.lower() == "h": return num \* 3600 return num \* 86400 def parse(self, text): in\_retrieval = False for raw in text.splitlines(): ln = raw.strip() if not ln or ln.startswith("#"): continue # block start/end if ln.startswith("retrieval"): in\_retrieval = True continue if in\_retrieval and ln.startswith("}"): in\_retrieval = False continue if in\_retrieval: # topk:X m = re.match(r'^topk\s\*:\s\*([0-9]+)$', ln) if m: self.retrieve\_topk = int(m.group(1)) continue #

synonyms:name=["a","b"] m = re.match(r'^synonyms\s\*:\s\*([A-Za-z0-9\_\-]+)\s\*=\s\*\[(.\*?)\]\s\*$', ln) if m: alias = m.group(1) lst = [x.strip().strip('"\'') for x in m.group(2).split(",") if x.strip()] self.synonyms[alias].extend([x for x in lst if x]) continue # synonyms support-thread=["check-in","mentor"] m = re.match(r'^synonyms\s+([A-Za-z0-9\_\-]+)\s\*=\s\*\[(.\*?)\]\s\*$', ln) if m: alias = m.group(1) lst = [x.strip().strip('"\'') for x in m.group(2).split(",") if x.strip()] self.synonyms[alias].extend([x for x in lst if x]) continue # ignore others inside block continue # expire m = re.match(r'^expire\s+(topic|tag|keyword):"([^"]+)"\s+after:([0-9]+)([dh])\s+action:(shield|remove)$', ln) if m: self.expire\_rules.append({ "kind": m.group(1), "key": m.group(2), "ttl": DSL.\_parse\_duration(m.group(3), m.group(4)), "action": m.group(5) }) continue # pin m = re.match(r'^pin\s+(topic|tag):"([^"]+)"\s+priority:([0-9.]+)$', ln) if m: self.pin\_rules.append({ "kind": m.group(1), "key": m.group(2), "prio": float(m.group(3)) }) continue # reinforce rule m = re.match(

r'^rule\s+on\s+event\s\*==\s\*"([^"]+)"(?:.\*?)->\s\*reinforce\s+(topic|tag):"([^"]+)"\s+by\s+([0-9.]+)(?:\s+cap:([0-9.]+))?(?:\s+cooldown:([0-9]+)h)?\s\*$', ln ) if m: self.reinforce\_rules.append({ "event": m.group(1), "kind": m.group(2), "key": m.group(3), "by": float(m.group(4)), "cap": float(m.group(5) or 1.0), "cooldown": int(m.group(6) or 0) \* 3600 }) continue # trust-forget m = re.match(r'^rule\s+on\s+trust\s\*<\s\*([0-9.]+)\s\*->\s\*forget\s+(topic|tag):"([^"]+)"', ln) if m: self.trust\_forget\_rules.append({ "threshold": float(m.group(1)), "kind": "topic" if m.group(2) is None else m.group(2), "key": m.group(3) if m.lastindex>=3 else "", "action": "forget" }) # robust parse alt form continue m = re.match(r'^rule\s+on\s+trust\s\*<\s\*([0-9.]+)\s\*->\s\*forget\s+(topic|tag):"([^"]+)"(?:.\*)$', ln) if m: self.trust\_forget\_rules.append({ "threshold": float(m.group(1)), "kind": m.group(2), "key": m.group(3), "action": "forget" }) continue return self # ---------- Core Engine ---------- class Engine: def \_\_init\_\_(self, memories, ctx, dsl: DSL): self.memories = [ensure\_defaults(dict(m)) for m in memories] self.ctx = ctx or {} self.dsl = dsl self.audit = [] # --- Rule applications --- def

apply\_expire(self): now = now\_ts(self.ctx) for r in self.dsl.expire\_rules: for m in self.memories: if match\_mem(m, r["kind"], r["key"]): age = now - ts\_of(m.get("timestamp")) if age >= r["ttl"]: if r["action"] == "remove": prev = m.get("weight", 1.0) m["weight"] = 0.0 self.audit.append({"type":"expire\_remove","id":m.get("id"),"prev\_weight":prev,"rule":r,"at":now}) else: if not m.get("shielded"): m["shielded"] = True self.audit.append({"type":"expire\_shield","id":m.get("id"),"rule":r,"at":now}) def apply\_trust\_forget(self): t = float(self.ctx.get("trust", 1.0)) for r in self.dsl.trust\_forget\_rules: if t < r["threshold"]: for m in self.memories: if match\_mem(m, r["kind"], r["key"]): prev = m.get("weight", 1.0) m["weight"] = 0.0 self.audit.append({"type":"trust\_forget","id":m.get("id"),"prev\_weight":prev,"rule":r,"at":now\_ts(self.ctx)}) def apply\_reinforce(self, event\_name=None): if not event\_name: return now = now\_ts(self.ctx) for r in self.dsl.reinforce\_rules: if r["event"] != event\_name: continue for m in self.memories: if match\_mem(m, r["kind"], r["key"]): last = float(m.get("last\_reinforced\_ts", 0)) if now - last < r["cooldown"]: continue prev =

m.get("weight", 1.0) m["weight"] = min(r["cap"], prev + r["by"]) m["last\_reinforced\_ts"] = now self.audit.append({"type":"reinforce","id":m.get("id"),"prev\_weight":prev,"new\_weight":m["weight"],"rule":r,"at":now}) # --- Retrieval --- def \_idf(self, docs): df = defaultdict(int) N = 0 for d in docs: N += 1 seen = set() for w in d.split(): w = w.strip().lower() if not w: continue if w not in seen: df[w]+=1; seen.add(w) idf = {} for w,c in df.items(): idf[w] = math.log(1.0 + (N/(1.0+c))) return idf def \_tfidf\_score(self, text, query\_terms, idf): if not text: return 0.0 words = [w.lower() for w in text.split()] if not words: return 0.0 tf = defaultdict(int) for w in words: tf[w]+=1 L = float(len(words)) s = 0.0 for q in query\_terms: ql = q.lower() s += (tf.get(ql,0)/L) \* idf.get(ql, math.log(1.0)) return s def \_expand\_query(self, q): parts = [p.strip() for p in q.split() if p.strip()] expanded = list(parts) for p in parts: if p in self.dsl.synonyms: expanded.extend(self.dsl.synonyms[p]) return expanded def retrieve(self, query, topk=None): topk = topk or self.dsl.retrieve\_topk # Build IDF on visible docs visible = [m for m in self.memories if not m.get("shielded") and m.get("weight",0)>0] idf = self.\_idf([m.get("text","") or "" for m in visible] or [""]) q\_terms = self.\_expand\_query(query or "") results = [] for m in

visible: base = float(m.get("weight",1.0)) \* float(max(0.0, m.get("trust",1.0))) tfidf = self.\_tfidf\_score(m.get("text","") or "", q\_terms, idf) if q\_terms else 0.0 pin\_boost = 0.0 for r in self.dsl.pin\_rules: if match\_mem(m, r["kind"], r["key"]): pin\_boost = max(pin\_boost, r["prio"]) score = base \* (1.0 + pin\_boost) + tfidf why = { "base\_weight": round(base,4), "tfidf": round(tfidf,4), "pin\_boost": round(pin\_boost,4), "final": round(score,4) } results.append((score, m, why)) results.sort(key=lambda x: x[0], reverse=True) out = [] for score, m, why in results[:topk]: out.append({ "id": m.get("id"), "topic": m.get("topic"), "tags": m.get("tags"), "weight": round(m.get("weight",1.0),4), "trust": round(m.get("trust",1.0),4), "timestamp": m.get("timestamp"), "text": m.get("text"), "score": round(score,4), "why": why }) return out # ---------- IO Helpers ---------- def load\_json(path, default): try: with open(path,"r",encoding="utf-8") as f: return json.load(f) except Exception: return default def write\_csv(path, rows, header): with open(path,"w",encoding="utf-8",newline="") as f: w = csv.writer(f) w.writerow(header) for r in rows: w.writerow(r) def main(): ap = argparse.ArgumentParser() sub = ap.add\_subparsers(dest="cmd") runp = sub.add\_parser("run", help="Apply rules, write before/after & audit CSVs")

runp.add\_argument("--mem", required=True) runp.add\_argument("--ctx", required=False, default="") runp.add\_argument("--dsl", required=True) runp.add\_argument("--audit", required=False, default="lethe\_audit.csv") runp.add\_argument("--before", required=False, default="lethe\_before.csv") runp.add\_argument("--after", required=False, default="lethe\_after.csv") runp.add\_argument("--event", required=False, default="") retp = sub.add\_parser("retrieve", help="Retrieve top memories for a query") retp.add\_argument("--mem", required=True) retp.add\_argument("--ctx", required=False, default="") retp.add\_argument("--dsl", required=True) retp.add\_argument("--query", required=False, default="") retp.add\_argument("--topk", required=False, type=int, default=0) args = ap.parse\_args() if not args.cmd: ap.print\_help(sys.stderr) sys.exit(1) memories = load\_json(args.mem, []) ctx = load\_json(args.ctx, {}) if getattr(args,"ctx",None) else {} with open(args.dsl,"r",encoding="utf-8") as f: dsl\_text = f.read() dsl = DSL().parse(dsl\_text) eng = Engine(memories, ctx, dsl) if args.cmd == "run": # before snapshot before\_rows = [] for m in eng.memories: before\_rows.append([m.get("id"), m.get("topic"), ";".join(m.get("tags",[]) or []), m.get("weight"), m.get("trust"), m.get("timestamp"), short(m.get("text", ""))]) write\_csv(args.before, before\_rows, ["id","topic","tags","weight","trust","timestamp","text"]) # apply rules eng.apply\_expire() eng.apply\_trust\_forget() if args.event: eng.apply\_reinforce(args.event) # after snapshot after\_rows = [] for m in

eng.memories: after\_rows.append([m.get("id"), m.get("topic"), ";".join(m.get("tags",[]) or []), m.get("weight"), m.get("trust"), m.get("timestamp"), "shielded" if m.get("shielded") else "", short(m.get("text",""))]) write\_csv(args.after, after\_rows, ["id","topic","tags","weight","trust","timestamp","flags","text"]) # audit audit\_rows = [] for a in eng.audit: audit\_rows.append([datetime.fromtimestamp(a.get("at",time.time())).isoformat(timespec="seconds"), a.get("type"), a.get("id"), json.dumps(a.get("rule",{}), ensure\_ascii=False), a.get("prev\_weight",""), a.get("new\_weight","")]) write\_csv(args.audit, audit\_rows, ["at","type","memory\_id","rule","prev\_weight","new\_weight"]) print(f"Done. Wrote {args.before}, {args.after}, {args.audit}") return if args.cmd == "retrieve": topk = args.topk or 0 results = eng.retrieve(args.query, topk=(topk or None)) print(json.dumps({"results": results}, ensure\_ascii=False, indent=2)) return if \_\_name\_\_ == "\_\_main\_\_": main()

# 부록 D. 감사 로그 원문(있을 경우)

type,memory\_id,topic,before,after,time,tag forget,1,ex-relationship,0.1,0.1,2025-08-22T02:51:27.023436, reinforce,2,,0.2,0.4,2025-08-22T02:51:27.023458,support-thread reinforce,5,,0.2,0.4,2025-08-22T02:51:27.023464,support-thread