ChatGPT

Engineering Solutions for Semantic Thermodynamics in Cognitive Systems

Introduction: Bringing Thermodynamics into Semantics

Semantic thermodynamics is an analogy for treating knowledge and symbols as if they carry “energy” that flows, decays, and interacts. In an intelligent agent, pieces of knowledge could have **symbolic energy** (a measure of current relevance or activation), which undergoes **semantic decay** over time if not reinforced. Conflicting ideas might create **contradiction buildup** – a pressure or tension in the system that needs resolution. Meanwhile, related or compatible ideas can exhibit **resonance activation**, mutually reinforcing each other’s activation. Finally, an agent may need an **ego-centric override** mechanism – a top-level control that can bias or override lower-level processes based on the agent’s core goals or self-preservation. Implementing these concepts requires careful architectural choices and engineering strategies. Below, we explore pragmatic solutions in existing cognitive frameworks, design patterns for real-time memory dynamics, and tools to build **AGI-like architectures** that embody these principles.

Active Cognitive Frameworks with Semantic Dynamics

Several cognitive architectures and AI frameworks have begun to integrate dynamic “mental” energies, decays, and conflict resolution. These systems, often in active research or prototype stages, demonstrate that semantic thermodynamics can be grounded in practical implementations:

- **OpenCog (Hyperon) – Attention Allocation:** OpenCog’s architecture uses an **AtomSpace** (a weighted hypergraph knowledge base) where each knowledge atom has **Short-Term Importance (STI)** and **Long-Term Importance (LTI)** values [1] [2]. These act as a form of symbolic energy indicating how urgent or valuable a piece of information is. An attention allocation subsystem (ECAN) continually updates these values. Unused knowledge gradually loses long-term importance; a dedicated “forgetting” process removes atoms whose LTI falls too low [3] [4] – an explicit implementation of semantic decay. Important concepts spread activation to related ones: atoms periodically **spread their STI/LTI to neighbors** via *Hebbian links*, analogous to neural spreading activation [5]. Notably, OpenCog also supports *inverse Hebbian links* (negative associations) so that activating one concept can suppress a contradictory or competing concept [5]. This provides a built-in way to model contradiction inhibition – if two nodes represent contradictory beliefs, linking them with an inverse relationship causes one to diminish when the other is highly active. OpenCog’s design includes a scheduler and “MindAgents” (processes) that manage these updates in real time [6] [7]. While still experimental, OpenCog has been used in prototypes (e.g. virtual agents and robotics) to manage attention, reasoning (via probabilistic logic), and goal-driven behavior within this energetic symbolic framework [8] [9].

- **OpenNARS – Adaptive Resource Allocation:** The Non-Axiomatic Reasoning System (NARS) and its open-source implementation (OpenNARS) explicitly address AI that must operate under **insufficient knowledge and resources**. OpenNARS attaches a **“budget”** to tasks and knowledge items, which includes *Priority* (short-term importance), *Durability* (decay rate), and *Quality* (long-term importance) [10]. These values are bounded between 0 and 1 and determine how much processing time and memory an item gets. As time passes or new information floods in, priorities decay according to their durability – implementing real-time memory decay and freeing attention for new, relevant inputs. Crucially, NARS's reasoning is **term-oriented and non-monotonic**; it can handle contradictions in a graded way. Each statement has a truth value (with uncertainty), and contradictory evidence adjusts this truth value rather than halting the system. There isn't a single "contradiction counter" in NARS; instead, **confidence in beliefs decreases when they encounter opposing evidence**, preventing any one contradiction from crashing the system. OpenNARS's attentional control mechanism stochastically selects tasks for inference based on priority, ensuring that highly activated or urgent tasks get serviced, while low-priority ones fade unless they become relevant again [11] [12]. This stochastic selection (via a structure called the *Bag*) ensures even lower-priority items occasionally get through – avoiding starvation and allowing novel or previously minor facts to gain relevance if circumstances change [13]. The result is a **controlled reasoning cycle** where memory items "age" and competitive attentional economics drive the system's focus, much like a thermodynamic system reaching equilibrium under resource constraints.

- **LIDA (Learning Intelligent Distribution Agent) – Decay and Global Workspace:** The LIDA cognitive architecture is inspired by **Global Workspace Theory** and models a human-like cognitive cycle. In LIDA, perceptual and memory nodes carry an activation level that decays if they are not reinforced [14]. For example, LIDA's *Transient Episodic Memory* stores recent events but **its traces naturally decay within hours or days** [15], mimicking the way humans remember yesterday's lunch but not details from weeks ago. Elements in working memory (the "workspace") compete for attention; attention codelets will push significant or novel information to a *global broadcast* (analogous to consciousness). This broadcast amplifies the chosen information's activation (a resonance effect), while other items not making the cut may fade. LIDA thus implements **limited persistence of knowledge and gradual forgetting**, preventing information overload. It also has mechanisms for **conflict resolution via coalition forming** – incompatible interpretations form separate coalitions of codelets, and only one coalition wins the global broadcast at a time, ensuring a coherent conscious content. Higher-level goal contexts in LIDA can bias this competition (e.g. an active goal context spreads activation to relevant perceptions or memories), functioning as an ego-centric override of sorts. In essence, LIDA shows how to architect a system where many processes run in parallel with spreading activation, but a **top-level attentional spotlight serializes the result**, enforcing consistency and goal alignment cycle by cycle.

- **MicroPsi 2 / Psi-Theory – Motivational Overrides:** MicroPsi is a cognitive architecture based on Dietrich Dörner's *Psi-theory*, implementing agents as **neuro-symbolic spreading activation networks** [16]. Nodes in MicroPsi represent concepts, situations, or procedures, and activation spreads along links to reflect context and relevance. These activations naturally dissipate over time if not renewed, providing a continuous decay model. What makes Psi-theory unique is its emphasis on drives and motivations: the agent is guided by a set of primary and secondary **drives (urges)** such as hunger, curiosity (uncertainty reduction), competency, social affiliation, etc. [17] [18]. All goals are linked to satisfying some drive, and **at any time multiple motives may be active, but only one becomes dominant** to decide behavior [19]. This dominant motive selection is effectively an ego-centric override mechanism. For instance, even if many perceptions or thoughts are active, a high-level drive like *reduce uncertainty* can take over if the agent encounters a contradiction or surprise in its environment. (Psi-theory explicitly treats **mismatches between expectation and reality as a**

source of "frustration" that increases the urgency of the uncertainty-reduction drive [18].) In MicroPsi's implementation, a strong drive state will bias the activation network — focusing processing on resolving the uncertainty or fulfilling the need, while other activations are suppressed. This is analogous to how a robot low on battery might override all other activities to find a charging station. Additionally, MicroPsi's spreading activation is modulated; links have parameters for how easily they transmit activation and can **weaken if seldom used** (a forgetting mechanism) or strengthen through learning. By combining a homeostatic drive model with spreading activation, MicroPsi provides a pragmatic template for systems that need to **balance multiple goals and resolve conflicts**: conflicting goals create internal pressure that is resolved by selecting one motive to pursue at a time, and the architecture ensures that decision propagates through the network (inhibiting actions or thoughts that don't serve the active goal).

- **Copycat and Fluid Analogies – Spreading Activation & Temperature:** Though not a full AGI architecture, Douglas Hofstadter's *Copycat* system is a classic example of **meaning dynamics** in a constrained domain (analogies with letter strings). Copycat consists of a *Slipnet* (a semantic network of concepts) and a *Coderack* (a pool of codelets that perform actions). Each concept node in the slipnet has an activation level that can increase when relevant to the current situation and decays over time if not used [20] [21]. When a concept's activation increases, it can spawn codelets that try to apply that concept in the Workspace (the current problem context); the *urgency* of these codelets is directly influenced by the concept's activation [22] [23]. This is a concrete implementation of **resonance activation**: the more a concept seems relevant (resonates with the situation), the more processing it attracts. Conversely, contradictory or irrelevant concepts remain at low activation and seldom get picked. Copycat also introduced a **"temperature"** metric as a measure of **contradiction or disorganization** in the workspace. Temperature rises when the system has made conflicting or partial interpretations (i.e., the overall state has low "perceived order") and falls as a coherent solution assembles [24] [25]. This temperature governs the stochastic choice of codelets: at high temperature (high conflict/uncertainty), the system explores more wildly (more randomness), whereas at low temperature it becomes more focused and deterministic [26]. In effect, Copycat's temperature is a **contradiction pressure gauge** – a high temperature signals unresolved tensions in the interpretation, prompting the program to keep trying alternative mappings until the tension (temperature) lowers. Engineers can take inspiration from this by introducing analogous "frustration" or "uncertainty" variables in their systems that tweak behavior when things aren't fitting well. While Copycat operates in a toy domain, the architectural principles (spreading activation, stochastic prioritized processing, and self-monitoring of coherence) are **directly translatable to engineering practices** in larger symbolic systems.

Each of these systems provides **concrete mechanisms** that realize aspects of semantic thermodynamics. They demonstrate that features like timed memory decay, activation spreading, and conflict resolution can be built into a runtime. Notably, many architectures combine symbolic and sub-symbolic (numeric) techniques – hence they are *hybrid*. For example, OpenCog uses numeric importance values and probabilistic truth values alongside symbolic logic; MicroPsi uses numeric activation in a semantic network; LIDA uses both connectionist-like activation and discrete representations. This hybrid approach is often key to implementing "energetic" concepts effectively.

Design Patterns for Decay, Conflict, and Activation

Implementing semantic thermodynamics in a custom system doesn't necessarily require adopting a whole cognitive architecture. We can identify common **design patterns and strategies** from the above frameworks that engineers can apply in their own systems:

- **Memory Decay (Forgetting Mechanisms):** To prevent a symbolic knowledge base from growing without bound and to prioritize recent or frequent information, introduce a decay process. A simple pattern is to give each knowledge unit (node, fact, rule) an **activation level or utility value** that diminishes over time (e.g. exponentially or linearly). For instance, in ACT-R (a well-known cognitive architecture), each memory chunk's base-level activation falls off as a power-law function of time since last use (simulating human memory decay). In practice, you can implement this by periodically multiplying activation values by a factor < 1, or subtracting a small amount at regular intervals. The system should remove or archive items whose activation falls below a threshold – similar to OpenCog's approach of removing low-LTI atoms [4]. It's often useful to have *two timescales* for decay: a short-term fast decay (to quickly drop fleeting irrelevant data) and a long-term slow decay (to eventually forget even persistent items unless they are refreshed). This dual timescale was explicitly used in OpenCog (STI vs LTI) and can be emulated by maintaining two values or a composite score for each item. **Best practice:** calibrate decay rates to the dynamics of your environment – e.g. in a chat-bot memory, user context facts might decay after minutes, whereas core knowledge might never decay. Also consider **contextual decay**: if the context shifts, you might rapidly decay or deactivate the previous context's details to avoid interference.

- **Spreading Activation and Resonance:** Represent associative relationships between knowledge items and propagate activation through those links to simulate resonance. If item A and B are related (by context, similarity, or logical link), then when A becomes highly active, B's activation should get a boost, and vice versa [5]. This can be done by iteratively updating connected nodes: for example, each cycle, set `activation_B += link_weight(A,B) * activation_A * some_factor`. Spreading activation enables **content-addressable retrieval** (related ideas come to mind together) and can detect resonance clusters (sets of items that keep activating each other). Use this to implement things like **context maintenance** (all items related to the current focus stay alive together) and **constraint satisfaction** (items that mutually support each other will sustain each other's activation, whereas isolated items will fade). A practical tool for this is a graph library (like NetworkX or Neo4j) where nodes have a numeric property for activation; you can write a small scheduler to perform activation spreading on the graph at fixed intervals or triggered by events. *Resonance activation* can also be hybridized with vector semantic similarity: e.g., if using word embeddings or concept embeddings, you can create links on-the-fly between symbols whose vector representations are close, then let activation spread over those links. This way, if a concept is active, semantically similar concepts (even if not explicitly linked by a rule) will get activation – a pragmatic way to achieve **fluid association**. Keep an eye on damping the spread (like electrical circuits, avoid positive feedback loops blowing up activation). Many architectures include a damping factor or limit the depth of spreading to manage this.

- **Contradiction Detection and Pressure Tracking:** In a symbolic system, contradictions can arise (e.g., one module concludes fact X, another concludes NOT X). Rather than immediately stopping with an error, an AGI-oriented design needs to handle this gracefully. One approach is to incorporate a **Truth Maintenance System (TMS)** or reason-maintenance module [27] [28]. A TMS keeps track of justifications for beliefs and can *retract* beliefs when they are no longer supported or when they conflict with higher-priority assertions. For example, if X and NOT X are both derived, the TMS would identify the assumptions leading to the conflict and retract one of them to restore consistency [29]. This is a **resolution mechanism** that prevents permanent contradiction, but it can be augmented with a "pressure" indicator: each time a contradiction is detected, increment a counter or raise a flag. If the same facts keep oscillating (being asserted and retracted repeatedly), the system could escalate this as a sign of an unresolved high-level inconsistency – possibly invoking a special conflict-resolution routine or seeking external input. Another method is to use **multi-valued or fuzzy logic** for truth values so that contradiction is a matter of degree. For instance, you could allow a proposition to have a truth value between 0 and 1 (or even a four-valued logic with "both true and false" state) to explicitly represent uncertainty or inconsistency. The "contradiction pressure" then might be the degree to which truth values are mid-range (indicating maximal uncertainty). In practice, a simpler heuristic works: count how many mutually exclusive facts are active at once, or how many times beliefs have flipped. Using Copycat's insight, you could map this to a single **"temperature" variable that increases with inconsistency** [24]. High temperature (or conflict count beyond a threshold) can trigger the system to enter an exploratory mode (try alternative assumptions, or ask a human for clarification in a human-AI system) – mirroring how Copycat increases randomness at high temp [26]. **Best practice:** log contradictions and their resolutions, as they often signal either gaps in knowledge or errors in perception. By tracking them, the system can learn over time which conflicts are recurrent and perhaps add new rules to avoid them. Also, contain the scope of contradictions via **contextual partitions** of knowledge: e.g. maintain multiple "micro-theories" or contexts so that contradictory beliefs can be true in different contexts (a strategy used in Cyc and multi-context systems). This way, global consistency is less of an issue; contradictions only matter if they appear in the same context.

- **Resonance and Pattern Completion:** Beyond simple pairwise activation spreading, consider implementing a **pattern completion** or resonance mechanism. For example, a **Hopfield network** (an energy-based recurrent neural net) will settle into an attractor that represents a consistent pattern, essentially completing partial patterns. Symbolically, one can simulate this by having a set of constraints or rules that define a "consistent state" and then iteratively adjusting activations to satisfy more constraints. A practical engineering solution is to use an **iterative relaxation algorithm**: Represent conflict as a numeric penalty (negative energy) and harmony as reward (positive reinforcement between compatible symbols). Then use an optimization method (like simulated annealing or constraint propagation) to maximize total harmony. This is akin to how Hopfield nets minimize an energy function. It might sound theoretical, but it can be applied in knowledge graph consistency checking or scheduling problems. The key idea is that **resonance = reaching a low-energy, high-consistency state**. During runtime, if certain pieces of knowledge strongly "want" to be true together (mutually supporting), the system should gravitate to a state where those are active and conflicting ones are inactive. Spreading activation with both excitatory and inhibitory links (as in OpenCog's Hebbian and inverse-Hebbian links) is one lightweight way to do this [5]. If a contradiction link (inhibitory) connects A and B, activating A will lower activation of B, and vice versa, eventually one wins out – **resolving the conflict by damping one side**. Engineers can design rule systems where contradictory rules produce an inhibitory relationship rather than an immediate error. This way the system can continue running with both pieces present, but one will naturally dominate in activation if it has more support or relevance. This "soft" approach to contradictions yields a more robust, flexible reasoning process.

- **Ego-Centric Override and Goal Bias:** In complex systems, it's useful to have a top-layer that can override or bias other processes based on global priorities (the "ego" of the system representing its core self or objectives). A common pattern is the **executive control module** or a set of highest-priority rules corresponding to survival or goal maintenance. For example, a robot architecture might use a *subsumption design* where low-level behaviors (wander, avoid obstacles) run continuously, but higher-level behaviors (e.g. urgent goal to recharge battery) can suppress the lower ones. In cognitive architectures, this can be implemented by assigning extra activation or priority to any structure related to a current top goal. We saw this in MicroPsi/Psi-theory: only one motive at a time is allowed to control behavior, ensuring coherence [19]. A practical approach is to maintain a global **goal list or drive meters**, and at each decision cycle, weight all candidate actions or inferences by how much they serve the top goal. If an inference result or perception contradicts the agent's *essential goal or self-model*, the ego-centric mechanism might either discount that information or route it for special handling (for example, require additional verification or involvement of a meta-reasoner). **Caution:** While ego-centric override can keep an agent focused and safe (e.g. not letting non-critical distractions derail a mission), if overdone it can lead to pathological behavior (ignoring important new data because it conflicts with a prior belief or goal). A best practice is to implement override as a bias in activation/priority rather than a hard filter. That is, make goal-related elements *10x more likely* to be chosen in attention, rather than *absolutely* preventing other elements. This still allows critical contradictory evidence to come through if it's strong or repeated, enabling the agent to eventually correct itself. Technically, ego-centric bias can be tuned by something as simple as adding a constant boost to activation of any item tagged as "goal-relevant," or using a separate gating network that multiplies activations by a goal matching score. Another tool is the concept of a **Global Workspace** (as in LIDA/GWT): many processes compete, but the global workspace will favor coalitions that align with current goals for broadcasting. In an implementation, you might have a dedicated thread or module that monitors all active processes and if a certain flag (like "emergency" or "critical goal unmet") is set, it can pause or preempt other processes. Modern real-time systems achieve similar control via *interrupts* or *priority scheduling*. In cognitive software, one can similarly schedule cognitive cycles such that "high ego priority" tasks run more frequently or immediately when triggered.

Tools and Technologies for Implementation

Building a custom system with these features can be complex. Fortunately, there are frameworks and libraries – from academic cognitive engines to general-purpose data structures – that can help:

- **OpenCog Hyperon:** The upcoming version of OpenCog (Hyperon) provides an updated Atomspace (graph database) and a language (MeTTa) for defining cognitive dynamics. It is designed to be extensible – you can define your own cognitive **MindAgents** that implement processes like decay or spreading activation on the Atomspace. This could be an ideal platform if you want a ready-made graph knowledge base with support for weighted links, pattern matching, and concurrency. For example, you could write a MindAgent that every second multiplies all STI values by 0.99 to decay them, and another that scans for any contradictory predicates and issues an inverse link between them to let them battle it out. OpenCog also comes with a probabilistic logic engine (PLN) that can handle uncertain inference, which is useful for reasoning under contradiction or with graded truth. The downside is that OpenCog is a research codebase – it may require significant learning and tweaking, but it embodies many of the "thermodynamic" semantics principles out of the box.

- **OpenNARS and NARX**: If your focus is more on reasoning and less on large knowledge graphs, OpenNARS (in Java) or NARX (an experimental NARS in C++) could be used. They provide a ready-made logic engine with built-in forgetting and attention control. You input premises and questions in a formal language and the system schedules tasks and derives answers under the hood. It will automatically drop low-priority tasks and recycle memory. You can observe how the budget mechanics work and adjust parameters like global forgetting rates. Since NARS is designed to be an **AGI control system**, it might also be integrated into robotics or agent simulations where sensor inputs are turned into NARS judgments. However, using NARS effectively requires understanding its Non-Axiomatic Logic syntax and semantics. It's powerful for research but not as straightforward as typical programming.

- **Cognitive Architecture Toolkits:** Academic projects like **ACT-R** and **Soar** provide simulation environments where you can implement toy versions of these mechanisms. For instance, ACT-R's code (in Common Lisp, with Python bindings available) allows you to define chunks and set parameters for their decay rates. It also has conflict resolution for production rules based on utility values. Soar (in C++/Java) has an episodic memory component and mechanisms for operator selection which could be adapted to include activation values. These are mainly used for modeling human experiments, but if you are prototyping an AGI-like agent, they offer a lot of built-in cognitive features. Soar's newer versions even explored approaches to forgetting and goal reprioritization, since long-running Soar agents needed a way to not accumulate endless learned rules [14].

- **Neuro-symbolic Platforms:** Consider tools like **Nengo (Neural Engineering Framework)** or **Spaun** (a large-scale neural model of cognition) if you want a more neural implementation of semantic dynamics. In Nengo, you can create populations of spiking neurons to represent concepts, where neural fatigue or passive current leak naturally implements decay, and recurrent connections implement spreading activation. While neural simulations can be heavy, Nengo is optimized and can even run on GPUs. It also lets you mix in simple logic or math operations. With Nengo, one could model a "working memory" that retains a concept for a short time unless refreshed by recurrent input, and associative memory that completes patterns (resonance) via attractor dynamics. The benefit here is you get time-continuous dynamics easily – the physics of the model handles decay and integration. The drawback is it's less interpretable and harder to directly script rules compared to symbolic systems. For a hybrid approach, one might use a neural net to manage activation levels and a symbolic reasoner to handle discrete logic, communicating between the two periodically.

- **Graph Databases and Streaming Processing:** If you want to implement from scratch in a modern software stack, you can leverage a graph database (like Neo4j, TigerGraph, or even an in-memory one like NetworkX for Python) to store your knowledge graph with properties. Use a background scheduler (could be as simple as a Python loop or an Apache Kafka Streams job) to update the graph continuously: e.g., decrease all "energy" properties by some amount, propagate activation along edges, and check for constraint violations. Some graph databases support triggers or periodic procedures – for example, Neo4j has the concept of scheduled jobs or you could use its APOC library to periodically run a query that decays relationships. Another useful technology is complex event processing frameworks (like Drools Fusion or Apache Flink). These allow you to define rules that operate on streams of data with time windows. For instance, you can represent facts as events that "expire" after a time unless renewed – a straightforward way to do forgetting. Drools (a rule engine in Java) can assign **salience (priority) to rules** and you can update salience at runtime. One could encode high-priority goal rules with permanently high salience (ego-centric overrides) and mundane rules with salience that drops if they haven't been triggered recently (simulating decay of reflexes). Additionally, Drools Fusion supports truth maintenance – when facts are retracted, any derived facts can be re-evaluated. By customizing the conflict resolution strategy of the rule engine (which rule fires when multiple are applicable), you can incorporate a mix of **specificity, recency, and priority**, which parallels a lot of what we discussed (e.g. prefer the newest relevant rule, unless an important goal rule is available to fire).

- **Hybrid Knowledge Repositories:** For extensibility, you might consider designing your system with both a **symbolic store and a subsymbolic store** that stay in sync. For example, maintain a traditional knowledge base (facts, rules, graph) *and* a vector-based embedding for each symbol. Tools like Facebook's *faiss* or Elasticsearch can serve vector similarity queries quickly. This means you can quickly find which symbols are semantically close to a given symbol to create dynamic resonance links. You could also train a simple neural network that predicts contradiction or conflict between facts (by embedding their text) – essentially giving a hint when two facts "sound" contradictory, even before logically proving it. While this veers into experimental territory, it shows how **modern AI techniques** can support a semantic thermodynamic system: the neural part provides a kind of intuition (energies, similarities), and the symbolic part provides explicit reasoning and discrete control.

Best Practices and Considerations

When engineering these mechanisms, keep a few general principles in mind:

- **Calibration and Adaptation:** The decay rates, activation thresholds, and conflict thresholds are critical parameters. In an extensible AGI-like system, these might need to adapt over time or differ per memory type. It's wise to make these parameters tunable at runtime or learnable. For instance, you might start with a fixed decay rate, but allow the system to adjust it based on memory load (if memory is filling up too fast, increase decay speed). Some research systems even tie decay to novelty: e.g., new information decays slowly at first to ensure it's considered, then faster if it proves irrelevant.

- **Monitoring and Debugging:** A dynamically changing knowledge base can be hard to debug. Build tools to introspect the state: e.g., periodically log the top-N active concepts (the "focus of attention" similar to OpenCog's attentional focus [30]) and the current conflict level (e.g., a "temperature" value). Visualization tools that show the graph with node glow intensity proportional to activation can be extremely helpful during development – you can literally see activation flows and verify that decay and resonance behave as expected. Many graph platforms allow data visualization, or you can output data to something like Gephi for a quick look.

- **Maintaining Stability:** When combining these processes, watch out for stability and oscillation. For example, if you have a very rapid decay but also strong resonance feedback, the system could oscillate (a concept activates its neighbors, then decays too fast and they deactivate, causing it to reactivate in a loop). Avoid extreme parameter values and consider adding friction like a refractory period (after a node has been highly active, enforce a short period where it can't spike again immediately). Borrowing from control theory or neural modeling can help here – effectively, you want the cognitive system's dynamics to eventually settle or cycle through useful patterns, not jitter chaotically. Copycat's temperature mechanism is one example of a stabilizer, gradually reducing randomness as the system "cools" into a solution [31] [25].

- **Extensibility and Modularity:** The goal is an **AGI-like architecture**, which implies you will likely add modules for perception, motor control, language, etc. Design the semantic thermodynamic core as a module that can interface with others. For instance, your contradiction detector should be able to take input from any knowledge source (vision module might assert something that contradicts what the language module said – this should be caught). Define clear protocols: e.g., all modules post their assertions to a global blackboard or graph, which then handles the spreading, decay, and conflict checks uniformly. This blackboard/graph could be considered the "psychic common room" of the agent's mind, and the thermodynamic rules keep that room orderly. Many architectures (LIDA, Global Workspace, Blackboard systems) follow this pattern and are naturally suited to insert these mechanisms.

- **Learning from Dynamics:** An advanced consideration is to make the system learn from its own dynamics. If a certain piece of knowledge keeps decaying and getting re-added repeatedly, maybe it should be stored more permanently (the system has learned it's actually important long-term). Conversely, if a particular contradiction arises over and over, the system could learn an exception or nuance to resolve it. Try to close the loop by having meta-rules that monitor the thermodynamic variables and adjust the knowledge base or strategies. For example, if "contradiction pressure" stays high in a domain, perhaps introduce a new conceptual distinction (split a concept into two to avoid clash) or label certain sources as more trustworthy. This moves the design from a static engineered system to one that **learns to self-regulate** its semantic energies, pushing closer to an autonomous cognitive system.

Conclusion

Implementing semantic thermodynamics in a cognitive system is an ambitious but increasingly feasible approach to achieving adaptive, resilient intelligence. By drawing on **proven architecture ideas** – like OpenCog's attention allocation, NARS's resource budgets, LIDA's global workspace with decaying memories, MicroPsi's motivated activation, and Copycat's dynamic temperature control – we can design AI systems that manage knowledge in a fluid, human-like way. The key is to combine symbolic representations with energetic metrics: numbers that capture recency, importance, and conflict, driving which symbols thrive and which fade. The engineering solutions range from using existing cognitive frameworks to integrating custom decay-and-spread processes into graph databases or rule engines. Always prioritize **pragmatism**: choose mechanisms that your system's scale and purpose can support in real-time. For instance, a game AI can adopt a lightweight activation network for NPC memory, whereas a research AGI platform might deploy a full Atomspace with ECAN. Focus on **extensibility** – your design should accommodate new modules and knowledge types without overhaul (e.g., the decay process should not be hardcoded to one kind of memory only).

By instilling concepts of symbolic energy, decay, and conflict resolution, we enable our AI agents to **forget gracefully, remember usefully, resolve their inner contradictions, and stay goal-directed** in the face of distraction. These qualities will be essential for any system aspiring to human-level autonomy and robustness. The marriage of symbolic reasoning with thermodynamic-inspired dynamics is a promising path forward in the quest for adaptive, general intelligence – one where meaning isn't static, but a lively, evolving process.

Sources: Semantic thermodynamics concepts and implementations were informed by cognitive architecture research and documented examples, including OpenCog's ECAN attention economy [3] [5], NARS's attentional control and decay parameters [10], LIDA's memory decay and attentional cycle [15], Hofstadter's Copycat analogy-making system [22] [24], and Dietrich Dörner's Psi-theory as implemented in MicroPsi [18] [19]. These sources exemplify the translation of theoretical principles into working systems that inspire the best practices described above.

---

[1] [2] [3] [4] [5] [6] [7] [8] [9] [30] Attention_Allocation_v8_ikle
https://agi-conf.org/2009/papers/paper_63.pdf

[10] [11] [12] [13] agi-conf.org
https://agi-conf.org/2020/wp-content/uploads/2020/06/AGI-20_paper_51.pdf

[14] (PDF) LIDA: A working model of cognition
https://www.researchgate.net/publication/28765131_LIDA_A_working_model_of_cognition

[15] D_07. LIDA - Deep Learning Bible - J. Artificial General Intelligence - Eng.
https://wikidocs.net/208189

[16] Microsoft Word - Draft Bach AGI 2012.docx
https://agi-conference.org/2012/wp-content/uploads/2012/12/paper_69.pdf

[17] [18] [19] Psi-theory - Wikipedia
https://en.wikipedia.org/wiki/Psi-theory

[20] [21] [22] [23] [24] [25] [26] [31] Hofstadterian Architecture
https://notes.fringeling.com/HofstadterianArchitecture/

[27] [28] [29] Reason maintenance - Wikipedia
https://en.wikipedia.org/wiki/Reason_maintenance