Indian Journal of Computer Science and Technology
https://www.doi.org/10.59256/indjcst.20260502087
Volume 5, Issue2 (May-August 2026), PP: 819-827.
www.indjcst.com
ISSN No: 2583-5300

Introducing Software Neurotechnology for Artificial Intelligence

Dr. Nitnem Singh Sodhi
Managing Director: Bharat Neurotech, Mental Health Specialist: Apollo Clinics Lucknow & Gorakhpur, Ex-Air Force Psychologist / Ex-UP Police Forensics Expert, Uttar Pradesh, India.

To Cite this Article: *Dr. Nitnem Singh Sodhi, "Introducing Software Neurotechnology for Artificial Intelligence", Indian Journal of Computer Science and Technology, Volume 05, Issue 02 (May-August 2026), PP: 819-827.*

Abstract: *The dominant paradigm for improving artificial intelligence systems over the last decade has been scale: larger models, trained on more data, with more compute, produce better performance. This paper describes a complementary and comparatively inexpensive paradigm, which we term Software Neurotechnology — a class of software layers that sit around a trained model and shape how it remembers, reasons, regulates itself, models the people and world it interacts with, and improves across repeated use, without altering the model's underlying weights. The term is a deliberate parallel to neurotechnology in the biomedical sense: just as a cochlear implant or a memory prosthetic augments a biological nervous system through an external interface rather than by rewiring the brain itself, these software layers augment an artificial one by restructuring its inputs, intermediate computation, and outputs. We present a twenty-layer framework organized into five categories — Memory, Reasoning & Planning, Self-Regulation, Interaction & World-Modeling, and Learning & Adaptation — describing the function of each layer, its relationship to existing empirical research, and its expected contribution to the practical, functional intelligence of a deployed system. We argue that this class of technique, largely orthogonal to model scale, represents an underexplored and comparatively cheap lever for improving AI systems, and we offer the framework as an organizing lens and a practical checklist for system builders.*

I. INTRODUCTION

A trained language model, considered purely as a next-token predictor, is not the same thing as a coherent, goal-directed problem solver. Two systems built on an identical set of weights can behave very differently in practice depending on what surrounds them: what state they are given at each turn, whether they are made to check their own output, whether they can consult external knowledge, and whether they retain anything from one interaction to the next. Much of the visible improvement in deployed AI systems over the past several years has come not from bigger models alone, but from an accumulating set of techniques that wrap a model in additional structure — memory objects, reasoning scaffolds, verification passes, and social models of the user. These techniques are usually described piecemeal, each under its own name, in its own paper, evaluated on its own benchmark.

This paper proposes a single frame for that whole class of technique: Software Neurotechnology. The name draws a deliberate analogy to biomedical neurotechnology, the field concerned with devices and interfaces that read from or write to a nervous system without altering its underlying biology — a cochlear implant does not rewire the auditory cortex; it restructures the signal reaching it. Similarly, the layers described here do not retrain or fine-tune a model's weights. They restructure what reaches the model (context, retrieved knowledge, a summary of the conversation so far), what happens between the model's passes (search, debate, critique), and what happens to what leaves it (verification, calibration, constraint checking). The underlying model is the nervous system; the layers are the technology built around it.

The immediate occasion for this paper was two such layers, identified independently as a starting point: a working-memory layer that maintains a small, explicitly structured state object for the current conversation rather than relying on the raw transcript, and a self-monitoring layer that checks a draft response against the original goal, the prior conversation, and any stated constraints before it is released. Both are simple to describe, both are cheap to implement relative to retraining, and both have reasonably strong empirical support in the surrounding literature, reviewed in the sections that follow. The contribution of this paper is to generalize from these two examples to a fuller taxonomy: twenty layers, organized into five functional categories, intended to cover most of the structural techniques currently used, individually, to make deployed AI systems behave more intelligently than the raw model alone would.

The paper proceeds as follows. Section 2 situates software neurotechnology relative to two older traditions: classical cognitive architectures, which first proposed that intelligence could be built from interacting functional modules, and the recent empirical literature on language-model agent scaffolding, which has independently rediscovered many of the same modules in a new substrate. Section 3 introduces the twenty-layer framework and its five categories. Sections 4 through 8 describe each layer in turn, category by category, drawing on existing research where it exists. Section 9 discusses how the layers interact and what an integrated architecture built from several of them might look like. Section 10 states the framework's limitations candidly. Section 11 concludes.

II.BACKGROUND AND RELATED WORK

The idea that intelligence might be assembled from a small set of interacting functional modules — rather than emerging from a single undifferentiated process — predates language models by decades. Newell and Simon's early work on human problem solving [18] framed cognition as search through a problem space, guided by heuristics and a working memory of the current state. This line of thinking was formalized into full cognitive architectures: SOAR [16], which represented all cognitive activity as search through problem spaces coordinated by a central working memory and a persistent production memory, and ACT-R [17], which distinguished declarative memory (facts) from procedural memory (skills) and coordinated them through a working-memory buffer system. Both architectures were built to be substrate-independent theories of the mind — general claims about what functional pieces intelligent behavior requires, largely independent of whatever mechanism implements pattern recognition underneath. That is the same spirit in which this paper is written, with a trained language model standing in for the underlying pattern-recognition substrate.

A second, much more recent tradition arrives at similar ground from the opposite direction. Rather than starting from a theory of mind and asking what substrate could implement it, the language-model agent literature started from a capable but narrow substrate — a frozen or lightly-tuned transformer — and asked what external structure would make it behave more like a coherent agent. Chain-of-thought prompting [4] showed that simply eliciting an explicit intermediate reasoning trace, without any change to the model, substantially improved performance on multi-step problems. ReAct [8] and Toolformer [9] showed that interleaving reasoning with calls to external tools extended a model's effective capability beyond what was encoded in its weights. MemGPT [1] showed that a model given an explicit, self-managed memory hierarchy could sustain coherent behavior far beyond its native context window. Self-Refine [10] and Reflexion [2] showed that a model critiquing and revising its own output, or its own past attempts, produced measurable quality gains with no additional training. Each of these results was published as a freestanding technique. Read together, they describe the independent, empirical rediscovery of something close to a cognitive architecture, built around large language models instead of production rules.

Software neurotechnology, as used in this paper, names that convergence and proposes a shared taxonomy for it. It is worth distinguishing the term from two neighboring ideas. It is not fine-tuning or reinforcement learning from human or AI feedback: those techniques change the weights themselves, and are consequently slow and expensive to iterate on, model-specific, and largely opaque about which change produced which behavioral effect. Constitutional AI [11] is a partial exception worth flagging explicitly — it uses a self-critique-and-revise mechanism, structurally identical to the self-monitoring layer described in Section 6.1, but uses it to generate training data rather than to gate a single output, showing that the same mechanism has value both as a runtime layer and as a training-time procedure. Nor is software neurotechnology simple prompt engineering, in the sense of a single well-worded instruction: the layers described here are persistent, structured, and often involve non-trivial computation — retrieval, branching search, multiple model calls — around a single forward pass, rather than a one-shot phrasing choice.

III.THE SOFTWARE NEUROTECHNOLOGY

The twenty layers below are grouped into five categories by the functional question each answers. Memory layers answer what does the system retain, across what timescale, and in what form. Reasoning & Planning layers answer how does the system manipulate what it retains to solve a problem it cannot solve in a single step. Self-Regulation layers answer how does the system check its own output before, during, or after producing it. Interaction & World-Modeling layers answer how does the system represent what is outside itself — the person it is talking to, and the environment it is acting on. Learning & Adaptation layers answer how does the system get better across repeated use, without any change to its underlying weights.

The two layers that motivated this paper appear inside this scheme rather than alongside it: working memory is Layer 1, and self-monitoring / critique-and-revise is Layer 9. Table 1 lists all twenty; Sections 4 through 8 describe each in turn.

Category	#	Layer	Core Function
I. Memory	1	Working Memory	Compact, structured state of the current task, updated every turn
I. Memory	2	Episodic / Long-Term Memory	Persistent record of past interactions, retrievable across sessions
I. Memory	3	Semantic Memory & Knowledge Grounding	Retrieval of external knowledge to ground responses in verifiable fact
I. Memory	4	Context Compression	Distillation of long histories into compact, decision-relevant summaries
II. Reasoning & Planning	5	Externalized Reasoning	Explicit intermediate reasoning steps before a final answer
II. Reasoning & Planning	6	Task Decomposition & Hierarchical Planning	Breaking large goals into ordered, tractable subgoals

II. Reasoning & Planning	7	Search & Exploration	Considering and comparing multiple candidate solution paths
II. Reasoning & Planning	8	Tool Use & Orchestration	Delegating subtasks to external calculators, APIs, or programs
III. Self-Regulation	9	Self-Monitoring / Critique-and-Revise	Checking a draft response against the goal before releasing it
III. Self-Regulation	10	Verification / Fact-Checking	Independently checking factual or logical claims in an output
III. Self-Regulation	11	Uncertainty Estimation & Calibration	Estimating and communicating confidence in a claim
III. Self-Regulation	12	Constraint & Guardrail Enforcement	Checking outputs against explicit and implicit rules
IV. Interaction & World-Modeling	13	Goal & Intent Tracking	Maintaining fidelity to the user's actual underlying objective
IV. Interaction & World-Modeling	14	Theory-of-Mind / Interlocutor Modeling	Modeling what the other party knows, wants, and feels
IV. Interaction & World-Modeling	15	Personalization / User-Preference Modeling	Persistent model of an individual's preferences and style
IV. Interaction & World-Modeling	16	World-State & Grounding	Maintaining a consistent model of relevant external state
V. Learning & Adaptation	17	Reflection & Lessons-Learned	Extracting reusable lessons from past successes and failures
V. Learning & Adaptation	18	Strategy Selection / Meta-Reasoning	Choosing which reasoning method fits the task at hand
V. Learning & Adaptation	19	Error Detection & Recovery	Noticing failure and triggering retry or repair
V. Learning & Adaptation	20	Multi-Agent Debate / Ensemble Cross-Checking	Using multiple model instances to critique one another

Table 1. The Software Neurotechnology framework: five categories, twenty layers.

A caution on the boundaries between layers, made explicit here so it need not be repeated twenty times: several of these divisions are functional rather than physical. A single implemented system might realize two adjacent layers (for instance, goal tracking and working memory) as one data structure with two logical fields. The framework separates them because they answer different questions and can in principle be improved, tested, or ablated independently, not because every implementation must keep them in physically separate components.

IV.MEMORY LAYERS

4.1 Working Memory

Working memory, as originally specified, is a compact, structured state object maintained per conversation — something like {current_goal, open_threads, key_constraints, recent_decisions} — recomputed or updated at every turn, rather than the full raw transcript. Its purpose is to keep an extended conversation coherent without relying purely on a large context window to hold everything ever said. The distinction matters: a raw transcript is undifferentiated, and a model reading it must re-derive, at every turn, what actually matters from what was merely said in passing. A working-memory object forces that judgment to be made explicitly and once, and carried forward.

The closest documented instance of this idea is MemGPT [1], which frames the problem in explicitly operating-systems terms: a language model has a fixed context window, analogous to physical RAM, and MemGPT manages movement of information between that fixed window and slower external storage tiers, in a manner directly modeled on virtual memory paging between fast and slow memory in a conventional operating system. Where MemGPT's contribution is chiefly about managing the boundary of what enters the context window at all, the working-memory layer as specified here is narrower and sits one level up: given that some information is in the active context, which small number of fields about the current task should be maintained in

a curated, explicitly labeled form, updated every turn, so that goal, constraints, and recent decisions do not have to be re-inferred from scratch each time. The two are complementary rather than competing: a system might use MemGPT-style tiered storage to decide what enters the window, and a working-memory object to decide what, within the window, is treated as authoritative current state.

4.2 Episodic / Long-Term Memory

Where working memory is scoped to a single ongoing interaction, episodic memory is scoped across interactions — a persistent record that lets a system "remember" a user, a problem, or a project over days or months, rather than starting fresh at every new session. Reflexion [2] provides a clean, minimal instance of this pattern in an agentic setting: after each attempt at a task, the agent converts scalar or binary outcome feedback into a short piece of reflective text, which is stored in an episodic memory buffer and supplied as additional context on the next attempt at a similar task — described by its authors as a form of verbal reinforcement learning, since it changes future behavior without changing any weights. MemGPT's archival memory tier, accessed on demand through explicit function calls rather than held permanently in context, is a second instance, oriented more toward retrieving specific past facts than toward accumulated lessons.

The functional requirement an episodic memory layer must satisfy is retrieval relevance: a system with an unbounded, unstructured log of everything that has ever happened gains little unless it can identify, from among a large history, the handful of past episodes actually relevant to the present moment. This is usually solved with some form of similarity search over stored episodes (commonly vector-based retrieval), making episodic memory the layer most directly dependent on the semantic-retrieval mechanism described next

.

4.3 Semantic Memory & Knowledge Grounding

A model's weights encode a great deal of factual knowledge, but that knowledge is frozen at training time, cannot easily be audited for its source, and cannot be updated without retraining. Retrieval-Augmented Generation [3] addresses this directly by combining two kinds of memory: parametric memory, held in a trained sequence-to-sequence model's weights, and non-parametric memory, an external, searchable index of documents consulted at generation time. The retrieved documents are supplied as additional context immediately before generation, letting the output be grounded in material that can be checked, cited, and updated independently of the model itself.

As a software-neurotechnology layer, semantic memory is the piece responsible for answering does the system need outside information for this response, and if so, what is the most relevant material available right now. Its value is largest exactly where a model's frozen internal knowledge is weakest: fast-changing facts, narrow or specialized domains, and anything the model needs to attribute to a specific, checkable source rather than assert from memory.

4.4 Context Compression

Left unmanaged, a transcript or a knowledge base grows without bound, eventually exceeding any practical context window and, well before that limit is reached, diluting a model's attention across material of wildly uneven relevance. A context-compression layer periodically distills accumulated history into a denser representation that preserves decision-relevant content while discarding detail that is no longer load-bearing. MemGPT's own overflow-handling mechanism is one instance: when the active context is full, it generates a new summary from the old summary plus the messages being evicted, an append-then-summarize strategy for keeping the working set bounded. More recent agentic-systems work on long-horizon tasks has proposed related "folding" mechanisms that compress the history of already-completed subgoals specifically to prevent that accumulated history from interfering with reasoning about what remains to be done.

Context compression is easy to confuse with working memory, since both reduce a large amount of material to a small amount, but the two operate on different objects. Working memory is a small, hand-specified set of fields about the current task, rebuilt fresh at every turn. Context compression operates on the raw history itself, producing a shorter version of the same kind of content, and is invoked periodically rather than every turn. A mature system typically needs both: compression to keep the underlying history tractable, and a working-memory object built from whatever the compressed history currently contains.

V. REASONING & PLANNING LAYERS

5.1 Externalized Reasoning

Chain-of-thought prompting [4] showed that eliciting a written sequence of intermediate reasoning steps, rather than requiring a model to jump directly to a final answer, substantially improves performance on arithmetic, commonsense, and symbolic reasoning tasks, with the effect most pronounced in larger models given only a handful of worked examples as demonstrations. No retraining is involved; the technique works purely by changing what the model is asked to produce before it produces a final answer. As a layer, externalized reasoning is best understood as a scratchpad: space in which the system is permitted, and encouraged, to work through a problem stepwise rather than commit to an answer in a single pass, in the same way a person is more reliable doing long division on paper than in their head.

5.2 Task Decomposition & Hierarchical Planning

Many goals cannot be solved in one step, or even one reasoning chain; they must first be broken into an ordered set of smaller, more tractable subgoals. A recent survey of planning in language-model agents [5] organizes this literature along several axes, task decomposition among them, and distinguishes single-path decomposition, where subgoals are chained in sequence, from tree-structured decomposition, where a goal branches into several subgoals pursued or evaluated in parallel. One representative method, ADaPT [6], decomposes a task only as needed: it first attempts a subtask directly, and recurses into finer-

grained decomposition only if the direct attempt fails, rather than exhaustively planning every task to the same depth regardless of whether the underlying model could already handle it in one step. That as-needed principle generalizes: a decomposition layer should be triggered by evidence of difficulty, not applied unconditionally, since unnecessary planning overhead has its own cost.

5.3 Search & Exploration

Standard generation commits to a single path token by token, with no way to look ahead, compare alternatives, or revisit an early choice that turns out to be wrong. Tree of Thoughts [7] generalizes chain-of-thought into an explicit tree: at each step the system generates several candidate next-thoughts, evaluates them, and uses that evaluation to decide which branches to continue, abandon, or backtrack from — a deliberate, look-ahead search process rather than a single greedy pass. The reported effect on tasks that specifically require this kind of lookahead is large: on a constrained arithmetic puzzle used as a benchmark, a single chain-of-thought pass solved a small fraction of instances, while the tree-structured search solved the large majority. The general lesson for a search-and-exploration layer is that some problems are better modeled as search over a space of partial solutions than as a single commitment to one line of reasoning, and that giving a system the ability to compare and discard candidates before committing captures real value on exactly those problems, at the cost of substantially more inference compute per query.

5.4 Tool Use & Orchestration

A model's weights are a poor place to store things that are cheap to compute exactly and expensive to memorize approximately — arithmetic, current data lookups, precise date calculations. Toolformer [9] showed that a model can learn, in a self-supervised way and without hand-labeled examples for each case, when to call an external tool such as a calculator, a search engine, a translator, or a calendar, what arguments to pass, and how to weave the result back into its own generation, producing marked improvements on tasks those tools are naturally suited for. ReAct [8] takes a complementary approach at the level of an interactive agent: it interleaves a reasoning trace with actions taken against an external environment, so that reasoning can update the agent's plan in light of what an action returned, and actions can be chosen in light of what the reasoning trace currently believes — letting the two processes correct each other over the course of a task rather than running independently. The general function of a tool-use layer is to extend a system's effective capability past whatever is encoded in its weights, by delegating well-defined subtasks to external, typically more reliable and more auditable, specialized systems, and treating the model's own role as deciding when delegation is warranted and how to use the result.

VI. SELF-REGULATION LAYERS

6.1 Self-Monitoring / Critique-and-Revise

The second layer specified at the outset of this project is a check, run before a response is released, against three questions: does this response actually satisfy the original goal, is it consistent with what was said earlier in the conversation, and does it violate any stated constraint. Self-Refine [10] operationalizes essentially this pattern using a single model in three roles: the same model first generates an initial output, then generates feedback on that output, then uses its own feedback to produce a revised output, repeating until a stopping criterion is met. No additional training, external feedback, or separate model is required, and the reported gains span a wide range of task types, from dialogue generation to mathematical reasoning to code readability, with the size of the gain varying considerably by task.

Constitutional AI [11] is worth discussing here even though it operates at training time rather than inference time, because the core mechanism is the same one: a model is shown its own response alongside a stated principle, asked to critique the response against that principle, and asked to produce a revised response, exactly the pattern of the self-monitoring layer. Constitutional AI's contribution was to show that responses produced this way can themselves become training data, so that the model is not only checked at output time but is gradually shaped, over training, to produce fewer outputs that would have failed the check in the first place. The relationship between the two is complementary: a self-monitoring layer catches problems in an already-trained model at the point of output; feeding its corrections back as training signal is one way to gradually reduce how often the layer needs to intervene.

6.2 Verification / Fact-Checking

Self-monitoring, as described above, checks a response holistically against the goal, the conversation, and stated constraints. Verification is narrower and more specific: it checks whether individual factual or logical claims within a response are actually true, typically by cross-referencing them against retrieved evidence (linking this layer to semantic memory, Section 4.3) or by having an independent pass, possibly a separate model call, evaluate a claim in isolation from the pressure of having just generated it. Multi-agent debate, discussed at length in Section 8.4, is one mechanism that produces a verification effect as a side benefit: when several model instances see and respond to each other's answers, an incorrect claim from one instance is exposed to challenge from the others, and empirical work on this approach reports improved factual accuracy and fewer confidently-stated errors relative to a single model answering alone [13]. The distinction between self-monitoring and verification is worth holding onto even though the two are often implemented together: a response can be perfectly consistent with the conversation and every stated constraint while still containing a factual error, and a response can be factually correct while still failing to address what the user actually asked. The two checks catch different failure modes and neither substitutes for the other.

6.3 Uncertainty Estimation & Calibration

A response is more useful, not less, when it is accompanied by an honest signal of how confident the system is in it. Kadavath et al. [12] studied this directly, finding that sufficiently large language models are reasonably well calibrated on questions posed in multiple-choice or true/false format, and that a model can be prompted to evaluate the probability that its own

already-generated answer is correct — a quantity the authors call P(True) — with calibration and accuracy improving further when the model is first shown several of its own independently generated samples before making that judgment. They also examined a related and harder quantity, P(IK), an estimate of whether the model is likely to know the answer to a question at all, prior to attempting one.

As a layer, uncertainty estimation sits downstream of a draft answer and produces a second output alongside it: not just a claim, but a confidence attached to that claim, which downstream layers (a verification pass deciding whether to double-check something) or the end user (deciding how much independent scrutiny a claim deserves) can use to allocate their own attention appropriately. Its absence is one of the more consequential gaps in current systems, which typically present all claims with the same uniform, confident tone regardless of how well-supported each one actually is.

6.4 Constraint & Guardrail Enforcement

Distinct again from the holistic quality check of self-monitoring, this layer performs narrow, explicit rule-checking: does the output stay under a stated word limit, follow a requested format, respect a safety or policy boundary, or avoid contradicting a factual commitment made earlier in the conversation. Because these checks are enumerable, they are well suited to being run as a discrete pass against a short, explicit list rather than derived freshly from the whole conversation each time — which is precisely what the key_constraints field of the working-memory layer (Section 4.1) is intended to supply. The practical value of keeping constraints in an explicit, structured field, rather than leaving them implicit in a long transcript, is largely realized through this layer: it is what makes constraint-checking a tractable, bounded operation instead of an open-ended re-reading of the entire conversation.

VII. INTERACTION & WORLD-MODELING LAYERS

7.1 Goal & Intent Tracking

A user's literal phrasing is sometimes an imperfect proxy for what they are actually trying to accomplish, particularly a few turns into a conversation that has moved on from where it started. A dedicated goal-and-intent-tracking layer maintains, and periodically re-derives, a model of what this person is actually trying to achieve, distinct from and feeding into the current_goal field of working memory. Where working memory simply holds whatever the current goal is currently believed to be, this layer is the mechanism responsible for noticing when that belief needs to be revised — for example, when a user's new message subtly redefines the task, or reveals that an earlier assumption about their goal was wrong — and for repairing the working-memory state accordingly rather than letting it silently go stale.

7.2 Theory-of-Mind / Interlocutor Modeling

Beyond tracking what a user wants, a system benefits from modeling what a user already knows, what would confuse them, and how they are likely to be feeling about the exchange — the general capacity psychologists call theory of mind, the attribution of mental states such as beliefs, knowledge, and intentions to another agent. A recent survey of this capacity in large language models [14] reviews both story-based evaluation benchmarks and methods proposed to strengthen it, and reports a mixed and evolving picture: language models have improved substantially on classic false-belief tasks as they have scaled, with one earlier study [15] finding a leading model of its time solving a majority of a standard false-belief battery, comparable to reported performance in six-year-old children on the same class of task in past developmental studies, while other evaluations find the capability to be brittle and highly sensitive to how a task is phrased. The honest summary is that this is an active and contested area of measurement, not a settled capability.

As a layer, interlocutor modeling is what lets a system tailor an explanation to what someone plausibly already knows rather than defaulting to one fixed register for everyone, notice signs of confusion rather than plow ahead, and adjust tone to the apparent emotional stakes of an exchange. It is distinct from personalization (7.3) in that it concerns the immediate, inferred mental state of the person in this conversation, rather than a standing profile built up over many past conversations.

7.3 Personalization / User-Preference Modeling

A personalization layer maintains a slowly-updated, cross-session profile of an individual's preferences — preferred level of explanation depth, domain background, recurring stylistic requests, standing constraints that hold across many separate conversations. It differs from episodic memory (4.2) in kind rather than degree: episodic memory is a log of specific past events, while a preference profile is a distilled, standing summary abstracted away from any single episode, closer to procedural knowledge about how to interact with this particular person than to a record of what happened.

This layer carries a design tension worth naming rather than glossing over. A system that adapts too readily to inferred preferences risks sycophancy — optimizing for what a user seems to want to hear rather than what is accurate or useful — and risks narrowing what it shows someone over time in a manner structurally similar to a filter bubble. A well-built personalization layer needs some explicit mechanism for distinguishing preferences about form (how something is explained) from preferences about substance (what is actually true), and for treating only the former as safe to optimize against without limit.

7.4 World-State & Grounding

Where semantic memory (4.3) retrieves general knowledge, a world-state layer maintains a live, accurate model of whatever specific external system the AI is currently operating over — the actual current contents of a document being edited, the actual current state of a codebase, the actual current position in a game or simulation — rather than relying on a possibly stale or hallucinated internal representation. ReAct [8] frames part of its contribution in exactly these terms, describing how interleaving actions with reasoning lets the model act to reason: querying an external environment specifically to keep its own internal state

grounded in current reality, rather than reasoning from an ungrounded and potentially outdated internal guess. The general requirement this layer satisfies is that anywhere a system is acting on a mutable external artifact, its model of that artifact's current state needs to be actively verified against the artifact itself, not merely inferred once and assumed to remain accurate.

VIII. LEARNING & ADAPTATION LAYERS

8.1 Reflection & Lessons-Learned

Reflexion [2] is the clearest documented instance of this layer: after an attempt at a task fails or succeeds, the agent converts that outcome into a short piece of verbal, textual reflection — what went wrong, what to try differently — which is stored and supplied as context on the next attempt, without any weight update. The authors describe this as a form of verbal reinforcement learning and report a substantial gain on a standard coding benchmark relative to a comparison model attempting the same tasks without this mechanism, illustrating that a training-free, purely-in-context feedback loop can produce gains of a magnitude usually associated with additional training.

Reflection differs from the within-conversation self-monitoring layer (6.1) along the same axis that separates episodic memory from working memory: self-monitoring fixes the current draft, within the current attempt; reflection operates across attempts, extracting a lesson intended to generalize to the next similar situation, whether that arrives moments later or in an entirely separate future session.

8.2 Strategy Selection / Meta-Reasoning

Not every task warrants every layer in this framework. A short factual question needs neither a branching tree search nor a multi-agent debate; a genuinely ambiguous, high-stakes planning problem might benefit from both. A strategy-selection layer is the piece responsible for that allocation decision: recognizing what kind of task is at hand and choosing which of the other nineteen layers, if any, are worth invoking, in effect deciding how much computation a given request deserves before spending it. Tree of Thoughts' own evaluation step, in which the model judges the promise of its own intermediate reasoning states before deciding whether to continue down a branch [7], is a narrow instance of this same general capacity, applied within a single search process rather than across the whole architecture. Generalizing it into its own layer treats task-appropriate resource allocation as a first-class concern rather than an incidental side effect of whichever techniques happen to be wired in by default.

8.3 Error Detection & Recovery

Distinct from a pre-output quality check, this layer operates during or immediately after an action is taken, particularly in agentic or tool-using settings: noticing that an API call returned an error, that executed code raised an exception, or that a plan step produced an outcome inconsistent with what was expected, and triggering an explicit recovery strategy — retry, backtrack to an earlier point in the plan, or surface the problem to the user — rather than silently continuing on a now-false assumption. ReAct's description of reasoning traces helping a model “handle exceptions” as it interacts with an environment [8] is an early, informal statement of exactly this requirement: that a system interleaving reasoning and action needs some mechanism for recognizing when an action did not produce the expected result, and for updating its plan in response rather than proceeding as if it had.

8.4 Multi-Agent Debate / Ensemble Cross-Checking

The last layer trades additional inference compute for a soft, distributed form of error correction. In the version studied by Du et al. [13], several independent instances of a language model each generate an initial answer to the same question; each instance is then shown the other instances' answers and reasoning, and generates a revised answer in light of them; this is repeated for several rounds before a final answer is taken. The reported effect spans both reasoning accuracy and factual reliability, with performance improving further as more agent instances and more debate rounds are used, at a roughly proportional increase in inference cost. The underlying mechanism is closely related to ensembling in classical machine learning — combining multiple independent estimates to reduce the error of any single one — conducted here in natural language, with agents reading and responding to each other's stated reasoning rather than only their final numeric outputs, which is what allows the process to catch reasoning errors and not merely disagreements about a final answer.

IX.DISCUSSION: LAYER INTERACTIONS AND TOWARDS INTEGRATED ARCHITECTURES

The twenty layers above are presented separately for clarity, but a real system rarely benefits from implementing them in isolation; several depend on or feed one another. Working memory's key_constraints field is what makes constraint enforcement (6.4) a tractable, bounded check rather than an open-ended re-reading of the transcript. Goal and intent tracking (7.1) is what keeps working memory's current_goal field from silently going stale. Episodic memory (4.2) supplies the raw material that reflection (8.1) distills into reusable lessons. Verification (6.2) and uncertainty estimation (6.3) both draw, where possible, on whatever semantic memory (4.3) can retrieve as external grounding. These dependencies suggest that the twenty layers are better understood as an interconnected system than as twenty independent modules to be bolted on in any order.

A second observation concerns cost. Not every task justifies all twenty layers, and running search, debate, and multiple verification passes on every request would be wasteful for the large majority of simple queries. This is precisely why strategy selection (8.2) is itself one of the twenty layers rather than an implicit property of the system as a whole: the decision of how many layers to invoke, and which ones, is properly understood as a layer in its own right, sitting logically prior to the others and governing how much of the rest of the architecture a given request actually activates.

It is also worth noting, without overstating it, that contemporary agentic AI system designs already tend to combine several pieces of this taxonomy in practice — typically some mix of memory, tool use, planning, and a reflection or critique step — which is one empirical hint that some such combination is already earning its keep, even though it is rarely organized under one explicit

account of why. Naming and structuring that combination, rather than proposing a wholly new technique, is the intended contribution of this framework.

Finally, the neurotechnology analogy that gives this framework its name is worth revisiting critically rather than only invoking it for flavor. The genuine parallel is that both a biomedical neural interface and a software layer of this kind operate on the boundary of a fixed underlying system rather than rewriting the system itself — restructuring a signal in, or an output out, without altering the substrate's own wiring. The parallel should not be pushed further than that: several of these layers, tree search and multi-agent debate especially, involve running multiple copies or multiple passes of the underlying model, which has no clean equivalent in a single biological nervous system. The term is offered as a framing device that makes an otherwise scattered set of techniques easier to think about together, not as a claim that the analogy holds in every particular.

X. LIMITATIONS AND OPEN QUESTIONS

Several limitations of this framework deserve to be stated plainly rather than left implicit.

Compute and latency cost. Most of these layers multiply the number of model calls or the amount of computation per response — self-monitoring at minimum doubles inference cost, multi-agent debate scales with the number of agents and rounds, and tree search scales with branching factor and depth. A production system must budget for this explicitly, and the strategy-selection layer (8.2) exists precisely because uniformly applying every layer to every request is not viable in practice.

Composability is not yet well established empirically. The individual mechanisms cited throughout this paper have each been evaluated on their own, typically on narrow benchmarks and typically in isolation from the other nineteen layers. Whether combining several layers produces gains that are additive, redundant, or occasionally net-negative — a critique layer, for instance, overriding a response that was already correct — is not well characterized across the full twenty-layer stack, and is a natural direction for future empirical work rather than something this paper claims to have resolved.

The boundaries between layers are functional, not physical, as already noted in Section 3, and reasonable practitioners could draw some of these divisions differently. The framework should be read as a useful organizing lens rather than a claim that these twenty categories are the unique or necessary decomposition of the problem.

Several of the interaction-modeling layers in particular — theory-of-mind modeling and personalization foremost among them — raise risks that a purely capability-focused framework does not by itself address: a system that models a user's mental state well is also a system that could, with different incentives, exploit that model manipulatively, and a personalization layer that optimizes too freely for engagement rather than accuracy risks the sycophancy and filter-bubble effects flagged in Section 7.3. These are genuine open problems in how such layers should be governed, and are outside the scope of a framework focused on describing what these layers do rather than how they should be constrained.

Finally, this paper's contribution is the organizing framework itself, built by generalizing from two specified starting layers to a fuller taxonomy grounded in existing published mechanisms; it is not a new experimental result, and the twenty-layer decomposition proposed here has not itself been validated as a unit, as distinct from the individual mechanisms it draws on, each of which has been independently studied by others.

XI.CONCLUSION

Model scale is one axis along which artificial intelligence systems improve. Software neurotechnology, as described here, is a second and largely orthogonal axis: a growing set of external layers — memory structures, reasoning scaffolds, self-regulation checks, models of the interlocutor and the world, and mechanisms for learning across episodes without retraining — that can be built around a fixed model to make its behavior more coherent, more reliable, and more genuinely useful in extended, goal-directed interaction. This paper organized twenty such layers, spanning five functional categories, into a single framework, grounding each in existing research where it exists and stating plainly where the framework itself remains untested as a whole.

The practical use of the framework is as a checklist. For any deployed AI system, it is possible to ask, layer by layer: which of these twenty pieces of structure are present, and is a specific, observed failure of coherence, reliability, or usefulness traceable to one that is missing. Future work should test combinations of these layers empirically rather than singly, develop cost-aware architectures that invoke only the layers a given task actually needs, and evaluate whether the twenty-layer decomposition proposed here holds up as a predictive tool, rather than only as a descriptive one, against systems built and measured against it directly.

REFERENCES

1. Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT: Towards LLMs as Operating Systems. arXiv:2310.08560.
2. Shinn, N., Cassano, F., Berman, E., Gopinath, A., Narasimhan, K., & Yao, S. (2023). Reflexion: Language Agents with Verbal Reinforcement Learning. Advances in Neural Information Processing Systems 36 (NeurIPS 2023).
3. Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., & Kiela, D. (2020). Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. Advances in Neural Information Processing Systems 33 (NeurIPS 2020).
4. Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-Thought Prompting Elicits Reasoning in Large Language Models. Advances in Neural Information Processing Systems 35 (NeurIPS 2022).
5. Huang, X., et al. (2024). Understanding the Planning of LLM Agents: A Survey. arXiv:2402.02716.
6. Prasad, A., et al. (2023). ADaPT: As-Needed Decomposition and Planning with Language Models. arXiv:2311.05772.
7. Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., & Narasimhan, K. (2023). Tree of Thoughts: Deliberate Problem Solving with Large Language Models. Advances in Neural Information Processing Systems 36 (NeurIPS 2023).
8. Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., & Cao, Y. (2023). ReAct: Synergizing Reasoning and Acting in Language Models. International Conference on Learning Representations (ICLR 2023).

9. Schick, T., Dwivedi-Yu, J., Dessì, R., Raileanu, R., Lomeli, M., Hambro, E., Zettlemoyer, L., Cancedda, N., & Scialom, T. (2023). Toolformer: Language Models Can Teach Themselves to Use Tools. Advances in Neural Information Processing Systems 36 (NeurIPS 2023).
10. Madaan, A., Tandon, N., Gupta, P., Hallinan, S., Gao, L., Wiegreffe, S., Alon, U., Dziri, N., Prabhumoye, S., Yang, Y., Gupta, S., Majumder, B. P., Hermann, K., Welleck, S., Yazdanbakhsh, A., & Clark, P. (2023). Self-Refine: Iterative Refinement with Self-Feedback. Advances in Neural Information Processing Systems 36 (NeurIPS 2023).
11. Bai, Y., et al. (2022). Constitutional AI: Harmlessness from AI Feedback. arXiv:2212.08073.
12. Kadavath, S., Conerly, T., Askell, A., Henighan, T., Drain, D., Perez, E., Schiefer, N., Hatfield-Dodds, Z., DasSarma, N., Tran-Johnson, E., et al. (2022). Language Models (Mostly) Know What They Know. arXiv:2207.05221.
13. Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., & Mordatch, I. (2023). Improving Factuality and Reasoning in Language Models through Multiagent Debate. arXiv:2305.14325.
14. Chen, R., Jiang, W., Qin, C., & Tan, C. (2025). Theory of Mind in Large Language Models: Assessment and Enhancement. Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL 2025).
15. Kosinski, M. (2023). Evaluating Large Language Models in Theory of Mind Tasks. arXiv:2302.02083.
16. Laird, J. E., Newell, A., & Rosenbloom, P. S. (1987). SOAR: An Architecture for General Intelligence. Artificial Intelligence, 33(1), 1–64.
17. Anderson, J. R. (1996). ACT: A Simple Theory of Complex Cognition. American Psychologist, 51(4), 355–365.
18. Newell, A., & Simon, H. A. (1972). Human Problem Solving. Prentice-Hall.