A Distributed Brain-Inspired AI Network with Suborgan Clustering for Evolutionary Emergent Intelligence

Abstract

We propose an updated design of the **Distributed Brain-Inspired AI Network (DBIAN)** framework, introducing a multi-region architecture that clusters neuron-like nodes into functional *suborgans*. Each suborgan acts as an independent "brain region" (e.g. hypothalamus, cortex) managing its own population of neuron-nodes, shared memory stores, and evolutionary controllers. Every neuron-node encapsulates a multimodal large language model (LLM) and a virtual machine (VM) for tool use, with connectivity to external Model Context Protocol (MCP) servers for data and resources. Suborgans specialize in distinct cognitive or control tasks (such as memory recall, planning, or hormone-like regulation) but collaborate via a global coworking interface analogous to a brain's integrative pathways. The framework enables rich interneuron communication using excitatory, inhibitory, and modulatory signals, and continuously evolves its network topology and parameters through genetic-inspired reproduction (two parent neurons "copulate" to produce four offspring with recombined and mutated genomes). We develop mathematical models for the communication dynamics, suborgan memory updates, and evolutionary reproduction processes. This paper situates the new DBIAN architecture in the context of related neuroevolutionary systems – including NEAT, EvoPrompt, and NeuEvo SNNs – and discusses how our biologically inspired design balances faithfulness to neural principles with necessary abstractions. We show how suborgan clustering and controlled neurogenesis aim to foster **emergent intelligence** through internal innovation and optimization. The proposed design, intended as a continuation of the DBIAN project, is presented with detailed component descriptions (main entity, suborgans, neuron-nodes, memory, communication, reproduction), updated architectural diagrams, example genome data structures, and comparisons to prior versions of DBIAN. We conclude with a discussion of the system's potential for open-ended learning, its advantages over earlier architectures, and remaining challenges and limitations.

Introduction

Artificial general intelligence (AGI) research increasingly looks to the human brain for architectural inspiration. The brain's intelligence arises from a **heterogeneous collection of specialized regions** (cortical and subcortical structures) that **operate in concert** to produce cognitionxmpro.comcourses.lumenlearning.com. Unlike monolithic AI models, the brain is massively distributed and integrates memory, perception, action, and regulation through collaborative subsystems. Recent survey work underscores that next-generation AI agents may

need a *modular design*, *self-improvement mechanisms*, and *multi-agent collaboration*, explicitly drawing parallels to brain organizationxmpro.com. In parallel, the success of Large Language Models has renewed interest in combining powerful learned models with cognitive architectures to approach human-like adaptive intelligencearxiv.orgxmpro.com. However, current agent architectures still fall short of brain-like generality and adaptabilityarxiv.org.

Distributed Brain-Inspired AI Network (DBIAN) was originally conceived as a step toward a brain-like AI system by distributing computation across many simple "neuron" agents (nodes) in a network. The initial DBIAN framework demonstrated the feasibility of a multi-node AI that loosely imitated neuronal interactions. Yet, that early design lacked the structural hierarchy and specialization that characterize biological brains – every node operated in a relatively flat, undifferentiated network, making it difficult to scale complexity or assign high-level functions. Moreover, adaptation in the original DBIAN was limited (e.g. basic learning rules), without an evolutionary mechanism to spur **internal innovation** beyond what was explicitly programmed.

In this work, we introduce a **significantly enhanced DBIAN architecture** that addresses these limitations by clustering nodes into *functional suborgans* and integrating an evolutionary algorithm at the core of the system's adaptation strategy. The **new DBIAN design** draws on biological analogy: suborgans resemble brain regions like the hypothalamus or hippocampus, each focusing on certain tasks and homeostatic roles, while an inter-suborgan communication interface resembles a *global workspace* or "coworking" hub facilitating information sharing across the whole artificial brain. Within each suborgan, neuron-like nodes now have substantial capabilities – each node runs a *multimodal LLM* instance (for natural language understanding and other modalities) along with a sandboxed VM to execute code or utilize tools, enabling complex reasoning and interaction. Nodes can also directly query external knowledge or services via standardized MCP servers, which act like a **"USB-C" interface connecting AI models to tools and data sources**modelcontextprotocol.iomedium.com. This design choice allows individual neurons to augment their built-in model with real-time information retrieval or computation, akin to how biological neurons rely on sensory organs and other systems to collect information.

A key novelty in our framework is the incorporation of **sexual reproduction and neuroevolution** within the agent. Inspired by genetic algorithms and neuroevolution techniques like NEATen.wikipedia.org, we allow neurons in each suborgan to "breed" new neurons, yielding offspring that inherit traits from two parents with random variation. This process, combined with a lifetime limit on each node's activity, creates a constantly evolving population of neural agents. Over time, the system can **optimize its configurations and behaviors** autonomously, seeking improved performance through variation and selection rather than requiring explicit retraining. The use of an evolutionary algorithm alongside gradient-based learning (internal to the LLMs) is in line with emerging research that **combines LLMs with evolutionary strategies for improved results**arxiv.orgarxiv.org. For example, EvoPrompt has shown that **LLMs coupled with evolutionary optimization can outperform static prompts** and discover solutions beyond human designarxiv.org. Our approach extends this idea from optimizing prompts to evolving entire neural agents within a larger cognitive system.

This paper is structured as a full-length academic study of the updated DBIAN framework. We first provide background on relevant biological and AI concepts that underlie our design (Section **Background**). We then review related work in brain-inspired architectures and neuroevolution (Section **Related Work**), positioning our contributions relative to known methods like NEAT, EvoPrompt, and the NeuEvo spiking neural network framework. Section **Architecture** details the DBIAN system's design, including the **main entity** (top-level agent), the definition and role of **suborgans**, the properties of **neuron-nodes**, the multi-layered **memory** structure, the **communication** protocols (signal types and coworking interface), and the **reproduction** mechanism. We provide updated diagrams to illustrate suborgan-node relationships, memory layers, and communication flows. Section **Methodology** describes how the system operates dynamically – how tasks are allocated, how suborgans coordinate, and how the evolutionary cycle is managed over time. In Section **Mathematical Models**, we formalize key aspects of the design with equations, including a model of neuron signal integration, a representation of the genetic crossover/mutation process, and a model of suborgan memory update and retention. We then discuss the **biological inspiration vs. abstraction** in our design, emphasizing why certain deviations from biology (such as neuron reproduction) are justified for functional reasons (Section **Discussion**). We also compare the new DBIAN with both its previous iteration and other architectures, highlighting improvements in modularity, adaptability, and emergent capability. Section **Limitations** candidly addresses the challenges and open issues with this approach (such as computational cost and complexity of analysis). Finally, Section **Conclusion** summarizes our contributions and outlines future directions for developing DBIAN into a robust platform for studying emergent intelligence in a brain-like AI network.

In summary, our **DBIAN with suborgan clustering** is a novel framework that marries brain-inspired structure with evolutionary learning principles. It aims to move closer to the vision of an AGI that not only mirrors the brain's modular intelligence, but also **evolves and innovates internally** to meet new challenges.

Background

Biological Inspiration: The human brain is composed of numerous specialized substructures (or "organs" within the brain) that handle different aspects of physiology and cognition. For example, the *hypothalamus–pituitary complex* serves as the hormone regulation center, translating neural signals into endocrine responses to maintain bodily homeostasis[courses.lumenlearning.com](courses.lumenlearning.com). The cortex is divided into regions for vision, auditory processing, motor planning, etc., and deeper structures like the hippocampus manage memory formation. Crucially, these suborgans do not operate in isolation: they are richly interconnected by neural pathways, allowing them to influence each other. Communication in the brain occurs via electrochemical signals of different types – **excitatory signals** that increase the likelihood of neuron firing, **inhibitory signals** that decrease it, and **neuromodulatory signals** (e.g. dopamine, serotonin) that adjust the overall context or learning parameters of networks. This complex interplay enables the brain's integrative function. Another key aspect is that the brain's structure arises through evolutionary processes (across species) and developmental processes (for each individual), although **adult neurons generally do not reproduce** themselves. In

contrast, adaptation happens via synaptic plasticity (learning) and some neurogenesis in limited areas. Our framework takes inspiration from the brain's **modular specialized design** and **rich signaling**, but extends it by introducing an explicitly evolutionary mechanism *within* the life of the agent (an abstraction that treats structural evolution as a continual process rather than an intergenerational one).

Brain-Inspired AI Architectures: The idea of dividing an AI into specialized modules is longstanding (e.g., Minsky's "Society of Mind" concept). Modern approaches like **Global Workspace Theory** suggest that specialized processors (modules) produce content that is broadcast on a global workspace for integration – a plausible model for conscious cognitive processingxmpro.com. Recent architectural proposals for AGI emphasize *modularity* and *multi-agent systems*. A 2025 technical survey by Liu *et al.* outlines a blueprint where cognitive modules (for memory, world-model, reasoning, etc.) are combined in a unified agent, which can also form teams of agents working togetherxmpro.com. This aligns with industry efforts to build AI systems composed of multiple LLM-based agents performing different roles cooperatively, rather than relying on a single giant model. These designs echo the distributed nature of the brain and have shown promise in solving complex tasks via division of labor and specializationxmpro.comxmpro.com.

Neuroevolution and Genetic Algorithms: Evolving the structure and parameters of neural networks through genetic algorithms has a rich history. **NeuroEvolution of Augmenting Topologies (NEAT)** is a seminal method that evolves artificial neural networks by gradually complexifying them – adding neurons and connections over generations – while using crossover and mutation to explore the space of topologiesen.wikipedia.org. NEAT introduced mechanisms like gene history tracking and speciation to preserve innovation and maintain diversity in the populationen.wikipedia.org. Variants and successors of NEAT (e.g. rtNEAT, HyperNEAT) have demonstrated evolving neural controllers for agents in simulated tasks in real-timeen.wikipedia.orgen.wikipedia.org. Neuroevolution tends to shine in scenarios where reward signals are sparse or where searching through architectures yields better solutions than backpropagation alone. Separately, **genetic algorithms** have also been applied outside of neural nets – for example, evolving text prompts for LLMs (as in EvoPrompt) or evolving program code. The principle of sexual reproduction (combining genetic material from two parents) and mutation (random changes) allows evolutionary systems to **explore creative solutions** that a single gradient descent path might not discover, occasionally resulting in *emergent behaviors*.

Model Context Protocol (MCP): Modern AI agents often need to interact with external tools, databases, or environments beyond their built-in knowledge. The Model Context Protocol (MCP) is an open protocol that standardizes how an AI model (especially an LLM) connects to various data sources and tools in a consistent waymodelcontextprotocol.io. One can think of MCP as a plugin interface providing a growing list of integrations (APIs, knowledge bases, calculators, etc.) that an AI agent can use on demandmedium.com. In DBIAN, we leverage MCP servers to give our neuron-nodes the ability to retrieve information or perform actions (for example, a neuron could query a factual database or call a web API) without hard-coding those capabilities. This is analogous to how real neurons rely on sense organs or actuators elsewhere in the body

– our artificial neurons rely on MCP endpoints to extend their functionality. By using MCP, we ensure each node can flexibly switch tools or data sources, and the overall system can incorporate new resources dynamically, which is essential for a long-lived evolving agent.

Prior DBIAN Framework: The original DBIAN (version 1.0) conceptualized an AI "brain" as a decentralized network of processing nodes (agents) loosely inspired by neurons. Each node could perform simple computations or host a small model, and nodes communicated messages to solve tasks collectively. However, that design treated all nodes uniformly and had no distinct higher-level organization. Learning in that system was limited to adjusting message-passing rules or weights; it did not include a mechanism for creating new nodes or morphing the network topology significantly during runtime. The updated framework described in this paper (which we might call **DBIAN 2.0**) builds upon that foundation but introduces *hierarchical organization (suborgans)* and *neuroevolution*. This represents a shift from a static multi-agent system to a **self-evolving cognitive architecture**.

In the following sections, we delve into the architecture and methods of DBIAN 2.0 in detail, after first surveying related work that informs its design.

Related Work

Research at the intersection of brain-inspired AI and evolutionary algorithms has produced a variety of frameworks and techniques relevant to DBIAN's design. Here we discuss several key areas and systems:

- **Neuroevolution (NEAT and descendants):** *Stanley and Miikkulainen's NEAT* algorithm demonstrated how genetic algorithms could evolve both the weights and **topology** of neural networks, starting from minimal structure and increasing complexity over generationsen.wikipedia.org. NEAT introduced **speciation** to maintain diverse niches in the population by grouping similar genomes, and a historical marking method to align genes during crossoveren.wikipedia.org. This addressed the problem of crossing over different network topologies in a meaningful way. *Real-time NEAT (rtNEAT)* later allowed continuous evolution in a running system: individuals have a limited lifespan and are replaced by offspring of high-fitness networks as they expire, enabling a form of ongoing adaptationen.wikipedia.org. This idea of assigning each agent a "lifetime" and asynchronously introducing new ones is directly analogous to our approach of giving DBIAN nodes a task-limited lifespan and continually breeding new nodes to replace aging ones. **HyperNEAT** extended NEAT to exploit geometric regularities by evolving an underlying function that encodes connectivity patterns, effectively generating large-scale neural structures with symmetry – useful for brain-like structures (e.g., retinotopic visual fields). While our DBIAN does not explicitly use HyperNEAT, the principle of generating structured connectivity resonates with our suborgan concept, where certain connectivity (like all memory nodes interconnecting with a shared memory module) is systematically defined. In comparison to these, DBIAN's evolution operates on a population of heterogeneous, **agent-like neurons** (each with internal complexity, like an LLM) rather

than simple neurons with a few weights. This raises new challenges for encoding a genome and defining mutation, but the high-level idea of *complexifying a network through evolutionary addition of nodes and links* is strongly influenced by NEAT.

- **Evolutionary Prompt/Program Optimization (EvoPrompt):** *EvoPrompt* is a recent framework that connects LLMs with evolutionary algorithms to optimize prompts for better task performancearxiv.org. It treats prompt strings as individuals in a population, uses the LLM itself to generate variations (mutation/crossover) of prompts, and selects the best-performing ones over iterationsarxiv.org. This approach yielded significantly improved prompts on many tasks and crucially demonstrated the synergy of combining **LLMs with evolutionary search**, leveraging the LLM's generative power with the EA's exploratory optimizationarxiv.org. For DBIAN, EvoPrompt is an inspirational example that *evolution can occur at the "meta" level above an LLM*, guiding it to better solutions without gradient training. In our case, each neuron-node's behavior (which could be thought of as partly determined by an internal prompt or configuration for its LLM) can be tuned via evolutionary means. Instead of evolving prompts externally, DBIAN evolves the **agents themselves** – but conceptually, the evolutionary process may alter a neuron's "prompt" (e.g., its internal role description or approach to tasks), hyperparameters like its creativity setting, or its tool-usage patterns. EvoPrompt underscores the importance of maintaining **human-readability and coherence** during evolution (prompts must remain intelligible). In DBIAN, we similarly must ensure that evolved neurons remain compatible with the system (for instance, offspring shouldn't degenerate into random gibberish LLMs – selection should favor those that still perform useful work). The EvoPrompt results indicate that evolutionary techniques can efficiently traverse discrete and non-differentiable search spaces, which is highly relevant since many aspects of our system (like discrete choices of which tools to use, or which connections to form) cannot be easily optimized by gradients.

- **NeuEvo for Spiking Neural Networks:** *NeuEvo* is a framework that evolves spiking neural network (SNN) architectures by incorporating a rich set of biologically inspired neural circuit motifsazoai.com. It combines **feedforward and feedback connections**, uses both excitatory and inhibitory neuron types, and applies unsupervised learning (STDP – spike-timing-dependent plasticity) for fine-tuning weightsazoai.comazoai.com. The evolution in NeuEvo is geared toward constructing **diverse neural circuits** that improve performance and efficiency on tasks like image recognition and reinforcement learning. Notably, NeuEvo's evolved networks include **different receptive field sizes and dense inter-cluster connections**, mirroring the complexity of biological neural circuitsazoai.com. This resonates with our approach of having different connection patterns and signal types in DBIAN (excitatory/inhibitory/modulatory links between neuron-nodes). One difference is that NeuEvo still deals with relatively homogeneous spiking neurons, whereas DBIAN's nodes are heterogeneous agents with complex internal models. However, the principle of evolving an interconnected network with specialized **neural populations** is very similar – we essentially scale that idea up to a cognitive architecture level. NeuEvo's demonstrated success in achieving state-of-the-art

performance in certain tasksazoai.com suggests that **guided evolution** can yield highly efficient and novel neural designs, which gives credence to our use of evolution to potentially discover emergent cognitive strategies within DBIAN. We also follow NeuEvo's example of mixing **unsupervised local learning rules with global evolutionary search** – in DBIAN, individual LLM nodes can still learn or fine-tune on the fly (analogy to STDP adjusting weights), while the global structure evolves.

- **Other Brain-Inspired Systems:** There have been various cognitive architectures and multi-agent systems influenced by the brain. For example, *OpenCog Hyperon* and related projects attempt to build AGI via interacting modules (symbolic and subsymbolic hybrid approaches). *MicroPsi* and *LEABRA* (O'Reilly's biologically based ANN framework) incorporate brain-like processes such as spreading activation and reinforcement learning with neuromodulators. While a full survey is beyond our scope, a common theme is emerging: a shift from monolithic models to **assemblies of specialized components** that can each be complex (like an LLM) but must work together. Our DBIAN framework, with its suborgans and coworking interface, contributes to this landscape by providing a concrete design that leverages modern AI tools (LLMs, MCP integrations) within a brain-like organizational structure. Unlike some cognitive architectures that are hand-designed with fixed modules, DBIAN adds an adaptive twist – the modules (suborgans and their neuron contents) can change and improve over time via evolutionary means. This is somewhat analogous to how **open-ended evolutionary systems** (like Karl Sims' evolving virtual creatures, or recent *open-endedness research*) continually produce novelty. We see DBIAN as bringing together ideas from **evolutionary AI** and **brain-inspired modular AI** into one unified framework.

In summary, DBIAN's design is informed by NEAT's network evolution, EvoPrompt's integration of evolution with LLMs, NeuEvo's multi-circuit brain-like approach for spiking nets, and the broad trend toward modular, collaborative AI agents. The combination of these influences yields a system that is, to our knowledge, unique: a distributed, evolving, brain-inspired agent network where each "neuron" is itself a learned model. In the next section, we describe the architecture of DBIAN in detail, illustrating how these concepts manifest in our system's components and interactions.

Architecture

The DBIAN architecture consists of a hierarchy of components organized in a brain-like manner. **Figure 1** provides a high-level overview of the architecture, showing multiple suborgans (brain region analogs) each containing a cluster of neuron-nodes, along with shared memory modules and evolutionary controllers within each suborgan. Suborgans communicate with each other through a central coworking interface, and individual neurons can interface with external tools/servers via MCP. We break down the architecture as follows:

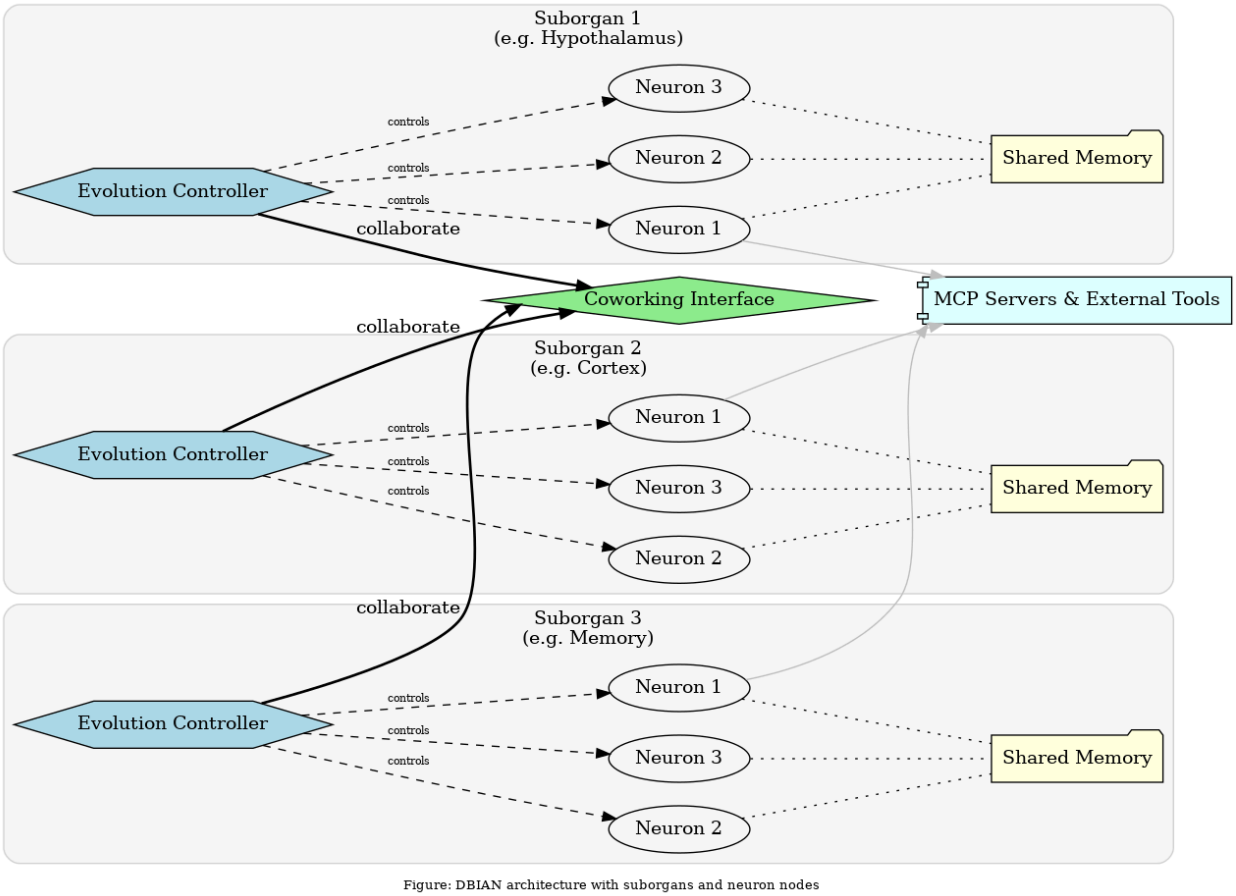

Figure: DBIAN architecture with suborgans and neuron nodes

Figure 1: The DBIAN architecture with suborgans and neuron nodes. Each suborgan (gray box) contains several neuron-nodes (oval), a shared memory (folder icon), and an evolution controller (blue hexagon). Neurons within a suborgan send/receive signals (dotted lines to shared memory, not shown: possible neuron-to-neuron links for excitatory/inhibitory signals within suborgan). The evolution controller monitors the suborgan's population and can trigger reproduction (not fully shown here) as well as send collaborative information via the Coworking Interface (green diamond). The coworking interface allows suborgans to exchange information or requests (thick black arrows). Some neurons are connected to external MCP servers/tools (right, light blue) enabling access to outside resources (gray arrows).

Main Entity (Top-Level Agent)

At the highest level, DBIAN can be seen as a single agent (or organism) composed of many parts. This *main entity* is responsible for interfacing with the external world or users. It receives **inputs** (which could be sensory data, user queries, etc.) and produces **outputs** (actions, answers, decisions). In implementation terms, the main entity could be a wrapper that delegates tasks to suborgans and then aggregates their responses. One may think of it as analogous to the *central nervous system* or simply the "brain" of an AI agent. However, unlike a simple monolithic brain, here the brain is explicitly constructed from sub-components. The main entity

holds references to all suborgans and maintains any global settings (for example, a global clock or global memory if needed). It also enforces top-level goals or drives of the system, ensuring that suborgans work towards the overall objectives.

In a practical scenario, if DBIAN is controlling a robot or a software agent, the main entity would receive a task (say, *"fetch item X from the warehouse"* or a complex query) and decide which suborgans need to engage. It might route perceptual data to a perception suborgan, planning requests to a planning suborgan, etc. It then uses the coworking interface to let these suborgans communicate and finally collects their outputs to form a coherent action plan or answer. The main entity may also implement global **clock cycles** or ticks if a synchronous operation is required, though much of DBIAN's internal processes are asynchronous by design (more like an event-driven brain than a lock-step program).

Suborgans

Suborgans are the primary functional modules in DBIAN. Each suborgan is conceptually akin to an independent brain region with a specialized role. For example, one suborgan might be designated for *Memory and Knowledge Retention* (like a hippocampus/neocortex analog), another for *Logical Reasoning and Planning* (prefrontal cortex analog), another for *Motivation and Task Prioritization* (analogous to limbic system or hypothalamus), and another for *Tool Interaction and Computation* (like a "digital motor cortex" driving external tools). The number and definition of suborgans can be configured depending on the application; they are essentially a way to impose structure on the swarm of neurons.

Each suborgan contains several key elements:

- A **Population of Neuron-Nodes:** These are the worker units (detailed in the next subsection) that carry out the suborgan's tasks. The suborgan might initialize with a certain number of neurons and can gain or lose neurons over time through reproduction and retirement. All neurons in a suborgan share the general functional focus of that suborgan, but may have varying strategies or sub-specialties that arise through evolution.

- **Shared Memory Store:** A suborgan provides a local shared memory mechanism that all its neurons can access. This can be thought of as a *blackboard or working memory* for that region. For instance, if the suborgan is "Memory," its shared memory might contain caches of facts or embeddings that neurons collectively maintain; if the suborgan is "Planning," the shared memory might hold the current plan steps or state information that multiple planning neurons work on. Technically, this could be implemented as a data structure (e.g., a dictionary or database) accessible via MCP or directly via an API that neurons call. The shared memory allows neurons to write intermediate results that others in the same suborgan can read, facilitating intra-suborgan cooperation. It also serves as a place to store the suborgan's persistent knowledge (like the best configurations found, or summary of past experiences).

- **Evolutionary Controller:** Each suborgan has a component responsible for overseeing evolutionary processes within that cluster. This is depicted as the “Evolution Controller” in Figure 1 (blue hexagon). Its role is to monitor neuron performance, decide when to trigger reproduction events, select parent candidates, and remove (retire) neurons that have reached end-of-life or are underperforming. The evolutionary controller may also adjust certain parameters of the suborgan (e.g., mutation rates, or turning on/off certain neurons via inhibitory signals) to guide the evolutionary process. Importantly, the evolution controller is *not* a single point of failure or an external operator; it can be implemented as just another node or a small utility within the suborgan. One could even imagine the evolutionary control being somewhat emergent or distributed, but for clarity we treat it as a distinct function. Biologically, this has no direct equivalent (since evolution in brains is not orchestrated by a “controller”), but it plays a role similar to a combination of neurogenesis regulators and a “manager” ensuring the population health of the suborgan.

In addition to these, suborgans have **interface handlers** for communication. Each suborgan connects to the central coworking interface (see below) for inter-suborgan messaging. Think of each suborgan like a *department in an organization*; it has its internal team (neurons), an internal bulletin board (memory), a manager (evolution controller), and a phone line to other departments (coworking interface).

Suborgans thus enforce a level of modularity. Neurons primarily interact within their suborgan (sharing memory, collaborating on tasks specific to that domain), and only higher-level results or requests go out via the coworking interface. This separation is useful for scalability – we can focus evolution and training within a suborgan without the full complexity of the entire system at once. It also allows assigning different metrics or selection pressures per suborgan if needed (for example, the memory suborgan might prioritize accuracy of recall, whereas a creativity suborgan might prioritize novelty).

Neuron-Nodes

Neuron-nodes are the atomic agents of the DBIAN network – each node corresponds to a single “neuron” in the brain analogy, but in practice it is far more powerful than a biological neuron. Every node encapsulates the following:

- An **internal model**, typically a Large Language Model (LLM) or a variant thereof (which could be multi-modal, hence capable of processing text, images, etc.). This internal model gives the neuron the ability to understand input, store knowledge, and generate output (e.g., a response or an action) in rich formats. Different neurons might use different underlying models or different parameter settings – part of what evolution might tune is which base model or fine-tuning a neuron uses. For instance, one neuron might be running a smaller, fast LLM to generate quick heuristics, while another uses a larger, more accurate LLM for detailed analysis.

- A **virtual machine (VM) or execution environment** that the neuron can use to carry out computations or actions. This VM could run code (Python, etc.) that the neuron writes, enabling it to solve logical or arithmetic problems, query databases, or orchestrate tool calls. In essence, each neuron can behave like an autonomous agent that can not only think (via its LLM) but also *act* (via code execution). The VM might be sandboxed for security and provided a certain toolkit (e.g., it might have an API to write to the suborgan's memory, or call certain allowed external tools).

- **Connectivity to MCP servers:** As mentioned, the neuron can connect to external resources. In practice, this means a neuron can issue requests to an MCP client which forwards them to appropriate MCP servers (for example, a server that provides web search, or a calculator, or a knowledge base query)[modelcontextprotocol.io](). The results come back into the neuron's context. By having standardized MCP connectivity, any neuron can, in principle, use any tool that is exposed via the MCP ecosystem, which greatly enhances capabilities. This is analogous to a neuron having "sensors" or "actuators" beyond itself. One could say these neurons are *tool-augmented LLM agents*. A neuron's genome (see Reproduction section) might encode preferences or settings for which tools it's particularly adept at using.

- **State and Properties:** A neuron maintains some internal state, such as:

    - A **neuron ID** and lineage information (who its "parents" are, generation number, etc.).

    - A count of tasks it has performed (to know when it has reached its lifespan limit).

    - Possibly its current "activation state" if we simulate activation (in our design, activation is more about messages and signals – see Communication – rather than a continuously running analog value).

    - **Connections/links** to other neurons: although communication is mediated by suborgans and coworking interface, we conceptualize that each neuron can have labeled links (excitatory, inhibitory, modulatory) to other specific neurons or suborgans. These could be represented as addresses or subscriptions: e.g., Neuron A might have an excitatory link to Neuron B meaning when A fires a certain message, B receives it as excitatory input. In implementation, this might be topics or message types rather than direct physical links.

Given their complexity, each neuron-node is more comparable to a small agent in a multi-agent system or a micro-service, rather than a single math neuron in a neural net. This is a deliberate design to leverage existing AI advances (LLMs, tool use) at the lowest level of our "brain." It's as if each neuron in a biological brain had a mini-brain of its own (which is biologically unrealistic, but computationally we can grant our neurons quite some power). The challenge then is to

coordinate these powerful neurons effectively – which is where the surrounding structure (suborgans, memory, signals) comes in.

Memory Layers and Shared Memory

Memory in DBIAN exists at multiple layers:

- **Local (Neuron) Memory:** Each neuron's LLM has an internal state or context (e.g., it can hold a conversation history or have some working memory through prompts). Additionally, during a task, a neuron can store variables or results in its VM environment. However, this memory is private to the neuron unless the neuron decides to share it. A neuron might, for instance, remember the outcome of the last tool it called, or keep a cache of recent facts it used.

- **Suborgan Shared Memory:** As noted, each suborgan provides a shared memory accessible to its members. This acts as a *short-to-medium term memory* for cooperation. For example, suppose the task is to plan a route on a map: multiple planning neurons might write candidate route segments or constraints to the shared memory so that others can refine or check them. The shared memory can also serve as a *repository of best practices* within that suborgan – e.g., the evolutionary controller might log which neuron (or which genome) was most successful on recent tasks, so that others can in some way use that information (or so that parents are chosen accordingly during reproduction).

- **Global Memory:** Though not explicitly mentioned in the problem statement, one can imagine a global memory accessible via the coworking interface or main entity. This would be analogous to very widely shared knowledge (like general facts, or a world model). If implemented, global memory would allow any suborgan to deposit information that might be useful system-wide. This could be implemented as another MCP server (like a central database).

- **Episodic vs Semantic Memory:** We can classify the content of memories. Some suborgans might maintain *episodic memory* (records of specific events or tasks the system solved, akin to an experience replay), while others hold *semantic memory* (refined knowledge distilled from experiences). For instance, a memory suborgan could over time build a knowledge graph or vector store of facts that any neuron can query (similar to how the human brain consolidates episodic events into semantic memory during sleep, etc.).

Memory management within a suborgan is partly handled by the evolutionary controller and partly by the neurons themselves. We plan for suborgans to track metadata about memory usage: which items were accessed frequently, which were ignored, how memory correlates with task success. When memory capacity is limited, suborgans may implement a forgetting mechanism (e.g., least-recently-used removal or a decay of rarely used entries). Mathematically, one can imagine each memory entry having a utility value that gets updated every time the entry

is used successfully, and entries falling below a threshold get pruned. (We will present a simple model in the Mathematical Models section.)

An important aspect is that suborgans also remember **the history of configurations** and **reproduction**. For example, a suborgan might keep a log: *"Genome X + Genome Y produced Genomes Z1..Z4 at time T, of which Z3 turned out highest performing"*. This evolutionary memory can guide future breeding (perhaps avoiding repeating combinations that failed, or preferring those that led to improvements). It also allows analysis of lineage – akin to a genealogy tree of the artificial neurons.

Communication and Coworking Interface

Neurons communicate using messages that are categorized as *excitatory*, *inhibitory*, or *modulatory*. In practice, all these are messages (e.g., text or data packets) but they carry a tag indicating their type and intended effect:

- **Excitatory message:** intended to stimulate action in the recipient. For instance, a neuron might send an excitatory signal to another with some content like "Idea A seems promising, expand on it." The receiving neuron, if it respects excitatory inputs, might incorporate that suggestion into its processing (like adding a bias towards that idea).

- **Inhibitory message:** intended to suppress or caution. E.g., "Don't consider option B, it's invalid." A neuron receiving this as inhibitory input might downweight or remove option B from its considerations.

- **Modulatory message:** intended to adjust the state or parameters of the recipient, rather than direct content. For example, a modulatory signal might say "Increase your cautiousness" which a neuron might interpret as lowering its generation temperature or being more risk-averse in decisions. Another modulatory signal might carry a reward/punishment value to reinforce learning ("The last action succeeded, remember that pattern"). These are analogous to neuromodulators in the brain that influence learning rates or excitement (like dopamine signaling reward, causing neurons to adjust synaptic strengths).

Within a suborgan, neurons could potentially have direct links to each other, but a simpler implementation is that they all share the suborgan's memory and possibly subscribe to certain message topics. We can imagine that excitatory/inhibitory signals are often directed to neurons of the same suborgan (short-range connections), whereas modulatory signals might be more diffuse and possibly cross suborgan boundaries (like a general alert or context reset).

Coworking Interface: This is the central hub through which suborgans interact. It's termed "coworking" to suggest a collaborative workspace. In implementation, it could be a publish/subscribe messaging bus or a shared blackboard where suborgans post requests and results. For example, if the Planning suborgan needs a piece of information that the Memory

suborgan holds (say a fact or an image), it can send a query via the coworking interface. The Memory suborgan's neurons monitoring the interface will pick it up, respond with the info, and the interface relays it back. The coworking interface thus routes messages between suborgans as needed.

The coworking interface can also handle conflict and integration: e.g., multiple suborgans might contribute different suggestions for an output, and a mechanism at the interface (or main entity) could reconcile them (like a "voting" or selection). It can be extended to allow **multi-suborgan meetings** – for instance, the interface could initiate a synchronized session where neurons from various suborgans come together to jointly solve something (analogous to cross-functional teams). However, in most cases, the interaction is asynchronous and mediated by messages.

An important design aspect is preventing communication overload: not every neuron should broadcast everything to all others. The interface can enforce that only salient or high-level information is shared across suborgans. Lower-level chatter remains within suborgans (which is similar to how in the brain, lots of neuron-to-neuron signaling is local, and only certain summary signals travel along long-range tracts or via neuromodulators broadly).

Reproduction and Lifespan of Nodes

DBIAN introduces a **genetic reproduction cycle** for neuron-nodes:

- Each neuron-node has a **lifespan** measured in number of tasks (or cycles) it can perform. This could be a fixed number (e.g., 100 tasks) or dynamically adjusted by the evolutionary controller. When the neuron has completed that many tasks, it "ages out" and is marked for retirement. It can finish its final task and then will be removed from the active set.

- Periodically or when certain conditions are met, the evolutionary controller of a suborgan will select two parent neurons from that suborgan to undergo reproduction. Selection might be fitness-based (e.g., pick the two that solved the most tasks successfully or the ones with highest performance metrics) or sometimes include a random component to maintain diversity. We may also incorporate speciation at the suborgan level: if neurons have very different "genomes", we might ensure mating occurs between compatible or suitably diverse pairs as appropriate (mirroring NEAT's speciation preserving innovationen.wikipedia.org).

- **Mating process:** The two parent neurons will combine their "genetic" information to produce offspring. The genome of a neuron encapsulates all information needed to instantiate a similar neuron. We will detail genome structure shortly, but it includes things like model parameters or identifiers, preferred tools, connection patterns, etc. A crossover operation is applied to the parent genomes: parts of the genome from parent A and parent B are mixed. For example, if we represent the genome as a set of genes (which could be binary strings, real-valued parameters, or symbolic configurations), we might randomly choose each gene for inheritance from either parenten.wikipedia.org.

We ensure that critical components are inherited in a compatible way – using techniques akin to NEAT's innovation numbers or aligning homologous genes could be considered, though in our case genomes might be more modular (e.g., one section for LLM hyperparameters, one for tool preferences, etc., which can be recombined).

- After crossover, each child genome is subjected to **mutation**: with some probability, small changes are introduced. A mutation could be flipping a bit (if a gene is binary), randomizing a parameter slightly (for continuous values), or altering a connection (e.g., adding/removing a link to a suborgan). Mutation rate and distribution are parameters managed by the evolutionary controller, possibly adapting over time.

- The result is a set of offspring genomes. According to the problem statement, we specifically create **4 new neurons from 2 parents** (so each mating yields 4 children). This is a design choice to amplify successful genetic combinations and also rapidly explore variations. It's akin to giving the offspring a "litter." Those 4 new neurons are then instantiated as active neurons in the suborgan. The parents might remain as well (especially if they still have lifespan left and continue to be useful), or we might retire the parents at this point to keep population size in check. There are multiple strategies here: one could maintain a steady population by retiring parents when offspring come, or allow some growth and then cull least fit later.

- Each new neuron is assigned a fresh lifespan counter and becomes part of the working population, ready to take on tasks.

Genome Structure: We now illustrate a simplified view of what a neuron's genome might look like, using a YAML-like format for clarity:

yaml
CopyEdit

```yaml
# Example of a neuron-node genome in YAML format
neuron_id: 47              # unique identifier
suborgan: "Memory"         # which suborgan it belongs to
model:
  base: "GPT-4-mini"       # identifier of base LLM model
  fine_tune: "memory-specialist-v2"  # optional fine-tuning ID
  temperature: 0.7         # decoding hyperparameter
modalities: ["text", "vision"]  # modalities the neuron can handle
tool_preferences:          # MCP tools this neuron is skilled at
  - name: "WebSearch"
    usage_bias: 0.9        # prefers to use this frequently
  - name: "Calculator"
    usage_bias: 0.5
```

```
connections:
  excitatory_targets: [51, 52]    # IDs of neurons it excites
  inhibitory_targets: [60]        # IDs of neurons it inhibits
  modulatory_targets: []          # no modulatory targets in this
example
lifespan_limit: 100         # tasks it can do before retirement
fitness_score: 0.85         # performance metric (computed during life)
parent_ids: [12, 15]        # IDs of parent neurons (for lineage
tracking)
mutation_rate: 0.01         # (could be uniform or gene-specific)
```

Listing 1: A hypothetical genome for a neuron node. In this example, the genome encodes the suborgan assignment, the model and hyperparameters (it's using a hypothetical "GPT-4-mini" model fine-tuned for memory tasks, with a certain creativity setting), the modalities (it can process text and vision), preferences for using two tools (with biases indicating how strongly it tends to use them), connections to other neurons by ID (perhaps within its suborgan or even cross-suborgan if allowed), its lifespan setting, a placeholder for fitness score (which is not part of the genome to inherit per se, but kept for selection purposes), and lineage info (who its parents were). Not all of these would necessarily be directly encoded for crossover (for instance, `fitness_score` might not carry over; it's more for selection). But parameters like model choice, fine-tune, and connections definitely would be subject to crossover/mutation.

During crossover, we might split the genome by sections. For example, one child might get the model parameters from parent A and tool preferences from parent B, while another child gets the opposite. Connections could be merged by taking the union or intersection of parent links plus some random additions/removals. Parent IDs of course would be set to the actual parents.

When a neuron is created from a genome, it loads the specified model (or a fresh copy of it) and configures itself accordingly. Note that large components like the base model weights are not mutated directly (we don't expect to evolve raw weights at the bit level – that would be infeasible given modern model sizes). Instead, mutation might toggle which base model is used from a small set, or adjust numeric hyperparameters. It's also possible to encode some **memory** or knowledge into the genome (for instance, the fine-tune name could imply inherited knowledge). In future implementations, we might even allow neurons to pass on some learned memory to offspring, a kind of Lamarckian inheritance (which is an interesting deviation from pure Darwinian evolution). For now, we assume inheritance is mostly of design/architecture/hyperparameters, not the exact learned weights (though offspring starting from same base model can learn similar knowledge during their life).

Lifespan and Retirement: The lifespan mechanism ensures continuous turnover in the population. A neuron's lifespan decrement could happen per task or per time unit. Retirement involves deregistering the neuron from the suborgan (so it no longer receives tasks or

messages) and freeing its resources. We may allow a retiring neuron to dump some of its knowledge into the shared memory before exiting (like “uploading” what it learned or summarizing its experiences), so that its contributions aren’t lost. This is analogous to how old members of a society might mentor or leave records for the next generation.

Putting It Together

When DBIAN is running, each suborgan will be concurrently solving sub-tasks with its neuron-nodes. They communicate locally (excit/inhib signals influencing which neurons take lead, modulatory signals adjusting parameters) and share memory. When needed, suborgans exchange information globally via the coworking interface. Meanwhile, in the background (or during idle times or after task completion), the evolutionary controller will spawn new neurons and retire old ones, based on performance logs and lifespans. Over time, we expect each suborgan’s population to become more efficient at its specialized task – akin to a micro-evolution optimizing memory-handling neurons in one suborgan, planners in another, etc. The global behavior of DBIAN emerges from the interplay of these evolving, specialized parts.

In the next section, we outline the methodology of how DBIAN operates step-by-step in a typical scenario, and then formalize aspects of the communication and evolution with mathematical models.

Methodology

We now describe how the DBIAN system functions dynamically. This includes how tasks are processed, how suborgans coordinate via the coworking interface, and how the evolutionary cycle is integrated into the agent’s operation.

Task Processing Workflow

1. **Task Ingestion:** The main DBIAN agent (main entity) receives a task or input. This could be a user query (in an interactive QA system), a sensory input frame (in a robotics context), or an internally generated goal. The main entity analyzes the task at a high level to determine which suborgans need to be involved. For example, if the task is “Describe how to tie a shoelace,” the main entity might engage the Memory suborgan (for any known instructions), the Planning/Reasoning suborgan (to formulate a clear step-by-step explanation), and a Language suborgan (if one exists specifically to refine output phrasing). It would not, say, engage a Vision suborgan since this task is purely textual.

2. **Suborgan Activation:** The main entity triggers those suborgans by sending an initial message or placing an entry in the coworking interface indicating the task context. Each relevant suborgan then activates its internal process. Activation might mean the suborgan’s neurons are given a goal or query related to the task. For instance, the Planning suborgan might get the prompt “Plan steps for tying a shoelace,” the Memory

suborgan might get “Recall any known procedures for shoelace tying,” etc.

3. **Intra-Suborgan Processing:** Within each active suborgan, neurons work (mostly in parallel) on their piece of the problem. How they divide work can vary:

    - In some cases, they may compete or cooperate. A possible approach is a **broadcast-and-competition**: the suborgan’s shared memory holds the current best solution attempt, and neurons attempt improvements. Excitatory signals can encourage certain neurons to contribute, while inhibitory signals can suppress redundant or unhelpful ideas.

    - For instance, one neuron in Planning might propose a set of steps and write it to shared memory. Another neuron reads it, sees a flaw, and via an inhibitory message suggests removing or altering a step. Yet another neuron might receive an excitatory cue to expand a particular step with more detail.

    - The suborgan’s evolutionary controller (or any supervisory logic) might ensure not all neurons activate at once to avoid chaos – possibly a scheduling mechanism or simply by the nature of signals some neurons wait until they are excited. This is analogous to how in a brainstorming session not everyone talks at once, but individuals contribute in turn.

    - Eventually, the suborgan arrives at an *output* for the current cycle: e.g., the Planning suborgan finalizes a sequence of steps.

4. **Inter-Suborgan Coordination:** Throughout the process, suborgans exchange information via the coworking interface:

    - The Memory suborgan might return a factual instruction sequence it remembered (“Loop the lace, pull through…”). This gets posted to the interface.

    - The Planning suborgan picks it up and integrates it into a coherent plan.

    - Perhaps a Language suborgan (if present) later takes the raw plan and rephrases it nicely.

    - If a conflict arises (say two suborgans propose different answers), the main entity or a designated arbitration suborgan reconciles it. This could be done by a simple rule (prefer plan with higher confidence) or by a joint discussion (one suborgan might ask another for justification via the interface).

    - The coworking interface may also handle timing – e.g., it can signal all suborgans when a final answer is needed, prompting them to wrap up.

5. **Task Completion:** The main entity collects the outputs from suborgans and composes the final result to output externally. In our example, it might take the refined step-by-step instructions from the interface (with language polishing done) and present that as the answer to the user. In a robotic action scenario, it might take a decision from a Decision suborgan and execute it on the robot.

During this task-focused processing, **neurons use MCP tools as needed**. For instance, if a neuron in the Memory suborgan didn't have the needed info on shoelace tying, it might call a WebSearch tool via MCP to find a wikiHow article, then parse it and share key points in memory. The use of tools is guided by the neuron's own policy (which can be evolved). One neuron might prefer using external knowledge, another might rely on internal knowledge. Over time, presumably the evolutionary process favors whichever approach yields better results for that suborgan's tasks (and likely it will be a combination of both).

Evolutionary Cycle Integration

The evolutionary mechanism operates in parallel with task processing. We design it to not disrupt tasks; ideally, evolution happens in the "background" or in between tasks:

- Each neuron has a lifespan counter. Each time it completes a task (or a unit of work contributing to a task), the counter decreases. The evolutionary controller monitors these.

- When a neuron's counter reaches zero, the controller flags it for retirement. It may allow it to finish any current contribution, then gracefully shut it down. Before removal, the neuron might dump any useful info to shared memory (e.g., "I found method X effective in solving Y"). That info can later be picked up by offspring or others.

- The controller decides when to initiate reproduction. This could be:

    - **Periodic:** e.g., after every N tasks handled by the suborgan, spawn some new neurons.

    - **On Retirement:** whenever a neuron retires, immediately breed new ones to replace it (this could even be two retirements triggering one mating to yield four children, thus expanding population by net +2).

    - **On Opportunity:** if two neurons have shown exceptional synergy or performance, the controller might proactively mate them even if they're not expired – to quickly propagate their traits. This is akin to seizing a good combination early.

- Selection of parents takes into account each neuron's *fitness*. Fitness can be a composite measure: number of tasks solved successfully, quality of contributions

(perhaps rated by the main entity or by outcome metrics), and efficiency (speed, resource usage). The highest fitness nodes are prime candidates. However, diversity is also important to avoid premature convergence. We might use a roulette wheel selection or tournament selection that sometimes picks a less fit but genetically distinct neuron. Also, to simulate speciation, the controller could cluster genomes by similarity and ensure mating happens within clusters or picks parents that are not too genetically distant (to avoid many broken offspring).

- Once parents are selected, the reproduction yields offspring as described. The offspring neurons are initialized and integrated into the suborgan:
    - They register with the suborgan's communication and memory systems.
    - They may start with an "orientation" period: e.g., read the shared memory to get up to speed on current context, or shadow a task to calibrate. Or they might be immediately thrown into the mix for a fresh task to see how they do.
    - Offspring could also potentially inherit some memory from parents – for example, the shared memory log could have entries like "Parent 12 learned that tool A is useful for Q". If the genome encodes parent IDs, the offspring might parse the memory for any notes from those parents to bootstrap themselves.
- The controller may impose a limit on total neurons. If population would grow too large, it might either reduce reproduction frequency or intentionally retire some neurons early (perhaps the poorest performers) to free slots. This keeps computational load manageable. It's similar to fixed population size in GA – some selection method to drop the weakest when new ones come in.

This evolutionary process is essentially a continuous online genetic algorithm operating within each suborgan. It bears similarity to rtNEAT's method where networks are continuously evaluated and replaced without halting the systemen.wikipedia.org. DBIAN thus maintains a sort of **meta-learning loop**: at one level, neurons (with their LLMs) learn and adapt within tasks (e.g., an LLM fine-tuning itself or updating its prompt strategies during interactions, analogous to synaptic plasticity), and at another level, less frequently, neurons themselves are replaced by new ones (analogous to a population evolving).

Example Scenario

To make this concrete, consider a scenario where DBIAN is an AI personal assistant that over time learns to better serve its user:

- Initially, suborgan A (Memory) has neurons that search the web for answers, and suborgan B (Reasoning) has neurons that try to reason through problems. Over many Q&A sessions, the controller notices two Reasoning neurons consistently provide

accurate answers and decides to breed them. The offspring get a mix of their traits: one offspring might inherit the tool-using propensity of parent1 and the logical style of parent2. These new neurons perhaps are even better, and the suborgan's performance on difficult questions improves.

- Another suborgan C (perhaps "Motivation/Prioritization") evolves neurons that manage what the assistant should do when multiple user requests come in. If a neuron makes poor choices (user is dissatisfied with what it prioritized), its fitness is low and eventually it's replaced by offspring of better neurons that handled priorities well. Over time, suborgan C's population converges to neurons that handle prioritization in a way tuned to the user's preferences (e.g., always address urgent queries first, etc.).

- Meanwhile, the Memory suborgan's shared memory accumulates a knowledge base of the user's favorite things. Neurons in Memory that effectively store and retrieve these preferences become highly valued. The evolutionary controller might keep those around longer or give them more offspring.

- If the user's needs change (say now the user asks more coding questions than before), the system can adapt: perhaps a new "Coding" suborgan could be spawned (this would be a larger structural evolution, which is an extension we could consider), or more likely the Reasoning suborgan evolves some neurons that are good at coding (maybe by spawning ones that use a Python tool extensively). There might be a mutation that adds a connection from a neuron to a new tool (e.g., a code interpreter), and if that helps answer coding questions, that neuron's fitness jumps and it propagates this trait.

Throughout this, the user hopefully experiences an assistant that is getting smarter and more attuned, without explicitly knowing that under the hood a population of "digital neurons" are competing and reproducing.

Modular Component Interactions

We emphasize how the components described in the Architecture section come together:

- The **Main Entity** orchestrates at high level but doesn't micromanage. It invokes suborgans and trusts them to do their job, and it handles final output composition.

- **Suborgans** manage internal complexity (like mini-agents within the agent). They expose an interface (via coworking) to request or provide info.

- **Neurons** carry out actual computations. They are largely autonomous – each decides when to send signals or write to memory based on its programming (which is partly from its LLM, partly from any coded policy).

- **Shared Memory** in each suborgan is accessed by neurons through reads/writes – perhaps via an API call in their VM (which could ensure concurrency control).

- **Coworking Interface** might be implemented as a message queue where suborgans subscribe to certain topics. When suborgan A needs something from B, it publishes a message “Request: X” tagged for B, and B’s listener picks it up. The interface also could log all inter-suborgan communication for analysis.

- **Evolutionary Controller** can be thought of as a background thread in each suborgan that wakes up at certain intervals, checks the state (fitness of neurons, any that died or retired), and triggers reproduction as needed. It updates a *Suborgan Memory of Evolution* (noted in architecture as tracking configurations and reproduction history). This could be stored in the shared memory or separately. For instance, after each breeding, it might store a record: (parent1_id, parent2_id, children_ids, context_of_breeding, initial performance metrics of children). Over time, it can analyze which parent pairs were most fruitful – akin to learning a good breeding strategy (perhaps certain combinations yield consistently good offspring, etc.).

The methodology ensures that **evolution and task-solving inform each other**: good performance on tasks increases reproductive success, and conversely the evolutionary changes produce (hopefully) even better task performance in the future. This feedback loop embodies an *optimization process* with respect to the goals the system is tasked with, albeit a very complex one. We rely on emergent evolution to handle aspects that are hard to explicitly program (like finding the optimal prompt or deciding which combination of tools yields the best result), by allowing variation and selection to gradually discover high-performing configurations.

In the following section on mathematical models, we formalize some parts of this process to give more precision to these descriptions.

Mathematical Models

In this section, we introduce mathematical formalisms for three core aspects of the DBIAN framework: **neural communication dynamics** (excitatory/inhibitory/modulatory signal integration), **genetic reproduction** (crossover and mutation of neuron genomes), and **suborgan memory management** (tracking and updating knowledge of efficient configurations). These models are abstractions intended to capture the essence of how the system operates, providing a theoretical foundation.

Communication Model

Each neuron-node can be seen as a processing unit that takes inputs (from other neurons or environment) and produces output signals. We denote by $O_i(t)$ the **output** of neuron i at time (or cycle) t. In an LLM-based neuron, $O_i(t)$ might represent a particular message or

result the neuron produces (e.g., the content it writes to shared memory or sends via coworking interface) during cycle t. For modeling purposes, we can consider $O_i(t)$ as a numeric activation or utility value representing the significance of neuron i's output at that time.

Neuron i receives inputs from potentially many other neurons. Let E_i be the set of neurons with excitatory connections to i, I_i the set with inhibitory connections to i, and M_i the set with modulatory connections to i. These sets are defined by the network's connectivity (which evolves over time). Correspondingly, each connection has a weight (or efficacy) that determines how strongly it influences i. Let $w_{j \to i}^+$ denote the weight of an excitatory connection from neuron j to i, $w_{k \to i}^-$ the weight of an inhibitory connection from k to i, and $w_{\ell \to i}^m$ the weight of a modulatory connection from ℓ to i.

We can then write the **net input** to neuron i at time t as:

Iinet(t)=∑j∈Eiwj→i+ Oj(t) − ∑k∈Iiwk→i− Ok(t) .I_i^{net}(t) = \sum_{j \in E_i} w_{j \to i}^+ \, O_j(t) \;-\; \sum_{k \in I_i} w_{k \to i}^- \, O_k(t) \,. Iinet(t)=j∈Ei∑wj→i+Oj(t)−k∈Ii∑wk→i−Ok(t).

This formula states that i sums up all excitatory inputs (each contributing their output times the excitatory weight) and subtracts all inhibitory inputs (each contributing their output times an inhibitory weight). Modulatory inputs are not summed in the same way; instead, they influence parameters of neuron i's response. We introduce a modulatory factor $M_i(t)$ that affects neuron i at time t:

Mi(t)=∑ℓ∈Miwℓ→im Oℓ(t) .M_i(t) = \sum_{\ell \in M_i} w_{\ell \to i}^m \, O_\ell(t) \,.Mi(t)=ℓ∈Mi∑wℓ→imOℓ(t).

Think of $M_i(t)$ as a context or gain factor. For instance, a positive $M_i(t)$ might lower i's firing threshold or increase its responsiveness, whereas a negative $M_i(t)$ could raise the threshold (making i less likely to act).

We then define neuron i's output based on these:

Oi(t)=F (Iinet(t); θi+Mi(t)) ,O_i(t) = F\!\Big(I_i^{net}(t); \; \theta_i + M_i(t) \Big) \,,Oi(t)=F(Iinet(t);θi+Mi(t)),

where $F(x; \theta)$ is a neuron activation function with parameter θ (which could be threshold or other bias). In a simple case, if we treat neuron i as a binary activator: $O_i(t) = 1$ if $I_i^{net}(t)$ exceeds $\theta_i + M_i(t)$, else 0. But in our LLM context, F could be more abstract – e.g., F could represent whether neuron i decides to contribute to the shared memory or remain silent. A high excitatory input might prompt it to speak up, whereas inhibitory input might silence it. The modulatory input $M_i(t)$ could dynamically adjust θ_i (the threshold) making it easier or harder to activate.

Alternatively, if we interpret $O_i(t)$ as not just binary but a continuous measure of contribution strength, F could be an identity (linear) or a sigmoid: $O_i(t) = \sigma(I_i^{net}(t) - (\theta_i + M_i(t)))$ for some sigmoid σ. For simplicity, one can assume each neuron has a

baseline threshold θ_i for activation and modulatory input effectively adds or subtracts from that threshold.

Interpretation:

- If neuron j sends an excitatory message with magnitude $O_j(t)$ and $w_{j\to i}^+$ is large, neuron i receives a strong push to activate (e.g., a planning neuron strongly suggesting another planning neuron to consider a path).

- If neuron k sends an inhibitory message, it provides a subtractive influence. For example, if $I_i^{net}(t)$ becomes negative or falls below threshold due to inhibitory inputs, neuron i might not produce output (like a memory neuron suppressing another's recall to avoid redundancy).

- A modulatory message (say from neuron ℓ) might not carry direct content but can cause $M_i(t)$ to be positive, effectively lowering θ_i. This could represent a signal like "be more active," resulting in neuron i being more likely to output even with smaller I_i^{net}. Conversely, a negative modulatory $M_i(t)$ raises the bar, telling i to be cautious or silent unless it has a very strong net excitatory input.

This model captures the essence of how multiple signals integrate. In a running DBIAN, these computations might not be explicit numeric calculations (continued)

This model captures the essence of how multiple signals integrate. In a running DBIAN system, these calculations might not literally occur as equations, but the behavior of the messaging system can be **interpreted** in these terms. For instance, a neuron's decision to output or stay silent can be seen as comparing an internal utility (based on received excitations/inhibitions) against a threshold modulated by context. The modulatory signals correspond to dynamic adjustments – e.g., a "focus" signal could globally lower thresholds in a suborgan to make neurons more eager to contribute at a critical moment, much like a surge of dopamine increasing neural activity in certain brain circuits.

This formalism also helps when designing or tuning the system. One could assign numerical weights to connections and implement a simplified **activation update rule** as above to decide which neurons get to write to shared memory next, etc. It provides a way to reason about stability (too many excitations could lead to everyone talking at once – akin to an epileptic seizure in a network sense – whereas too many inhibitions could silence the system). Tuning the balance of excitatory and inhibitory influences is crucial for coherent behavior, just as in biological neural networks a balance is needed for stable dynamic[azoai.com](azoai.com)】.

Reproduction (Genetic Algorithm) Model

We model each neuron-node's *genome* as a vector of genes or parameters: $G = (g_1, g_2, \ldots, g_L)$. The length L and contents of this vector depend on our genome design (as

illustrated in the YAML example earlier). For simplicity, assume we have a fixed-length encoding for key traits (some genes might be binary flags for tool preferences, some might be numeric hyperparameters, others might be identifiers for model type, etc.). The **fitness** of a neuron i, denoted F_i, is a measure of its performance over its lifetime (higher is better). This could be a weighted sum of task success rates, contribution quality, etc., accumulated until it reproduces or dies.

When the evolutionary controller selects two parent neurons A and B for mating, they have genomes G^A and G^B. We perform **crossover** to produce an intermediate offspring genome G^{AB}. A simple crossover strategy is a gene-wise random mix (sometimes called uniform crossover):

gkAB={gkA,with probability 0.5,gkB,with probability 0.5,for each gene k=1…L.g^{AB}_k = \begin{cases} g^A_k, & \text{with probability } 0.5, \\ g^B_k, & \text{with probability } 0.5, \end{cases} \qquad \text{for each gene } k = 1 \ldots L.gkAB={gkA,gkB,with probability 0.5,with probability 0.5,for each gene k=1…L.

In other words, the child randomly inherits each gene from one of the two parents with equal chance. (More sophisticated crossovers could be used, such as 1-point or 2-point crossover if genes have an order, or blending for numeric values, but uniform gives a baseline.) This yields a combined genome G^{AB} that is a mix of A and B.

Next, we apply **mutation** to G^{AB}. Define a mutation operator $\mu(\cdot)$ that perturbs a gene. For each gene k:

- With probability p_{mut} (the mutation rate), we set $g^{child}_k = \mu(g^{AB}_k)$.

- With probability $1 - p_{\text{mut}}$, we leave it $g^{child}_k = g^{AB}_k$.

The mutation $\mu(g)$ could be defined in various ways depending on gene type:

- If g is binary or a categorical choice, $\mu(g)$ might flip the bit or switch to a different category (possibly with some bias).

- If g is numeric (like 0.7 for a temperature setting), $\mu(g)$ might add a small random delta drawn from a distribution (e.g., Gaussian with mean 0 and small standard deviation) and clamp the result within valid bounds.

- If g is an identifier (like which base model to use), $\mu(g)$ might randomly choose a different valid identifier from the set (simulating a random mutation that swaps the model).

We can incorporate *gene-specific mutation rates* if needed; for now assume a uniform p_{mut} for all genes for simplicity.

The outcome of crossover+mutation is one **offspring genome** G^{child}. To produce four offspring, we can repeat this process four times (each time may result in slightly different mixes and mutations due to randomness). Denote the four children's genomes as G^1, G^2, G^3, G^4. These are then instantiated as new neuron-nodes in the suborgan.

If parents A and B have high fitness, typically their children will start with a good combination of traits, but mutation ensures new variations also appear. Some offspring might not perform well (that's expected; not all children of great parents are great). The idea is that by generating multiple offspring, we increase chances that at least one or two will exceed the parents by getting a lucky combination of their complementary strengths. The **selection pressure** in the system comes from the fact that only neurons which survive and perform (i.e., avoid retirement and perhaps are chosen again as parents) will propagate their genes further. Over many generations, one would expect convergence to sets of genes that are well-suited to the tasks the suborgan handles.

To formalize selection a bit: when picking parents, one could use a probability proportional to fitness. For example, if suborgan S has population ${1,2,\ldots,N}$ with fitness $F_1, \ldots, F_N$, we might pick parent A with probability $F_A/\sum_{i\in S} F_i$ (roulette selection), and similarly for B (ensuring $B \neq A$). Alternatively, a **tournament selection** of size k could be used: pick k random neurons and choose the best among them as A; repeat for B. These methods are standard in genetic algorithms to probabilistically favor the fitter individuals while still giving others a chanc[en.wikipedia.org]. DBIAN's evolutionary controller can adopt any such method. Speciation, if implemented, means we would restrict selection to within a species (e.g., within clusters of similar genomes) to maintain diversit[en.wikipedia.org]. In practice, suborgans themselves could be seen as species, or if a suborgan's internal population diverges (some neurons with very different strategies), the controller might avoid mating those too-different ones and rather mate like with like or ensure innovation survives by not always mating only the top two (which might be very similar).

Continuous Evolution: We do not have discrete generations in DBIAN. The above process happens whenever triggered by the controller. We can model it in discrete steps for analysis: say every T tasks or time units, a mating event happens. Over time, let's denote by $P(t)$ the set of active genomes at "epoch" t. The evolutionary process is:

$$P(t+1)=(P(t)\setminus D(t)) \cup B(t), P(t+1) = \big(P(t) \setminus D(t) \big) \;\cup\; B(t), P(t+1)=(P(t)\setminus D(t)) \cup B(t),$$

where $D(t)$ are the genomes of neurons that died/retired in the interval (removed from population) and $B(t)$ are the genomes of newly born neurons in that interval. The size of $B(t)$ depends on how many reproduction events happened. In steady-state, one might have the number of births equal the number of deaths to keep $|P|$ roughly constant. If each reproduction yields 4 and we retire 2 (the parents or others) at that time, net +2, we might later

retire more to compensate. Over a long run, assume population size N is regulated to stay around some target.

We can also consider an evolving **gene pool** in the suborgan memory. Let $\mathcal{G}(t)$ represent the multiset of all gene values present in the population at time t. We could track frequencies of gene variants (alleles) and examine how they change – akin to population genetics. For example, if gene k is "uses tool X or not", we might see the allele "uses tool X" go from 30% to 80% frequency over time if tool X turns out very useful. This is another way to quantify the system's adaptation.

Suborgan Memory and Knowledge Management Model

Each suborgan's memory and logs help guide its internal processes and future evolution. We outline a few elements that suborgans track and how they can be modeled:

- **Efficient Configuration Records:** A configuration could refer to a particular arrangement or state of the suborgan's network that proved effective. For instance, "Neuron 5 strongly active, Neuron 7 suppressed, using Tool A frequently" might be a configuration pattern that yielded high reward. The suborgan can store a set of top configurations $C_{\text{best}} = {c_1, c_2, \ldots, c_m}$ with associated performance metrics $U(c_i)$ (utility scores). When new outcomes occur, the suborgan updates this:
    - If a new configuration c_{new} achieved utility $U(c_{\text{new}})$, and if $U(c_{\text{new}})$ is higher than the lowest in C_{best}, it can be added (possibly evicting the worst). This is like maintaining a **priority queue** of best scenarios.
    - The utility $U(c)$ might be defined as a weighted sum of accuracy, speed, resource cost, etc., depending on what "efficient" means (likely task reward minus some penalty for resources).
    - This record can inform decisions: e.g., the evolutionary controller might attempt to recreate or sustain a known good configuration by favoring certain genes or by sending modulatory signals to push the current state towards a recorded good state.
- **Communication Flow Patterns:** The suborgan can log sequences or graphs of message exchanges that led to successful task completion. For example, it might note that when neuron X excited Y and then Y inhibited Z, the outcome was good. Over time, it can derive a pattern like "X → Y (excitatory) followed by Y ⇐ Z (inhibitory) is a useful motif." We could formalize this as a frequency count of n-gram patterns in communication:
    - Let's define a "communication event" as $(i \xrightarrow{\text{sig}} j)$ meaning i sent a signal of type sig to j. We can count occurrences of sequences like

$(i \to j), (j \to k)$ and measure correlation with success.

    - The memory might store a weighted directed graph G_{comm} where nodes are neuron IDs (or roles) and edge $(i \to j)$ has a weight representing how often that communication (with whatever signal type) appears in successful episodes minus unsuccessful ones. This effectively learns a subnetwork of *useful communication links*. If G_{comm} shows a particularly strong link, the controller might ensure future neurons preserve that link (when breeding, favor keeping that connection).

    - Conversely, if some communication flow is frequently associated with failures (e.g., a certain neuron always sends a distracting excitatory signal that leads others astray), that might appear as a negative weight, and the controller could decide to remove or weaken that link (perhaps by mutating the offending gene that causes it).

- **Tool Use Statistics:** For each external tool (or MCP server function) T, suborgan memory can maintain stats like:

    - n_T = number of times tool T was invoked by any neuron.

    - s_T = number of successful outcomes from using T (success could be defined as tool returned useful data that contributed to solving task).

    - We can define an empirical success probability $\hat{p}_T = \frac{s_T + \alpha}{n_T + \alpha + \beta}$, using a Beta prior (α,β) to avoid zero-division (this is a Bayesian estimate). For instance, with $\alpha=\beta=1$ (uniform prior), $\hat{p}_T = \frac{s_T + 1}{n_T + 2}$.

    - The suborgan can prefer tools with higher $\hat{p}_T$. If a neuron has a choice of calling either a web search or a local database, and the memory knows web search success rate is 90% vs database 50%, it might excite neurons that tend to use web search.

    - Tools also have a usage cost; memory might store average time or tokens used per tool call, enabling a trade-off analysis (a fast but slightly less accurate tool might be better under time constraints).

    - When mutation introduces a new tool usage (a gene flips from “not using T” to “using T”), the initial success might be low, but if it occasionally yields a big improvement, the stats will update and could prove its worth.

- **Reproductive History:** The suborgan logs each reproduction event: $(id_A, id_B) \to (id_1, id_2, id_3, id_4)$ at time t, along with any immediate observations like parents’

fitness and initial children performance. Over time, genealogical trees or lineages can be constructed. This can be analyzed to detect phenomena like:

- *Lineage success:* e.g., “most of the current high-performers descend from ancestor node 7 about 3 generations back”. The controller could then examine what was special about that ancestor (maybe a particular gene) and ensure it’s preserved.

- *Inbreeding or loss of diversity:* if the log shows many events are between closely related individuals, genetic diversity might be shrinking. The controller might then decide to introduce a random new genome or force mating between dissimilar ones (somewhat akin to injecting novel DNA or encouraging exploration).

- *Mutation impact:* by comparing child performance to parents, the log can estimate which mutation types tended to be beneficial. For example, it might find that whenever the “model type” gene mutated to a larger model, fitness improved, suggesting that gene has not converged and maybe pushing further could help (at cost of resources). Or it might find too high a mutation rate causing many failures, and thus dynamically adjust p_{mut} downward for stability.

From a modeling perspective, we can represent the *knowledge in memory* as a set of variables updated over time. For instance:

- Let $Q(c)$ be the recorded utility of configuration c, updated as a running average when c (or something similar to c) occurs.

- Let W_{ij} be the weight for communication $i \to j$, updated as $W_{ij} \leftarrow W_{ij} + \eta ,(r - \bar{r})$ whenever $i \to j$ happened in a task with outcome r (reward) relative to average $\bar{r}$ – a kind of Hebbian credit assignment for communications.

- Let $\hat{p}_T$ update via Bayesian posterior after each use of tool T as mentioned.

- Let ΔF records for reproduction: e.g. for each event, store $F_{\text{child avg}} - \max(F_A, F_B)$, which is the difference between average child fitness and parent fitness. A positive average indicates sexual reproduction often yields better-than-parents (on average), which is good; a negative would indicate maybe something’s wrong (like epistasis issues or disruptive crossover). We could aggregate these stats.

These mathematical notions inform how one might implement learning at the suborgan level on top of evolution. The **memory acts as a critic**, guiding the evolution with additional feedback beyond raw fitness. In future iterations, this could lead to meta-learning: the suborgan might

adjust how it selects parents or what mutations to favor, essentially evolving the evolutionary strategy itself (though that is beyond our current scope).

Discussion

The DBIAN framework described here is a bold synthesis of biological inspiration and computational pragmatism. In this section, we discuss how **biological principles** influenced the design, where we **deviated or abstracted away from biology** and why, and what the implications are for achieving the goals of emergent intelligence and internal innovation. We also compare the updated DBIAN to its previous incarnation and to related systems to highlight improvements.

Biological Inspiration vs. Abstraction

We took strong inspiration from the *architecture of the brain*: modular organization, specialized regions, and multi-channel communication. Suborgans in DBIAN explicitly mirror brain regions with distinct functionxmpro.com】, which should facilitate interpretability (each cluster of neurons has a known role) and efficiency (task-specific optimization). We also mirrored the idea of **excitatory and inhibitory signaling** – crucial for brains to regulate activity – by allowing neurons to send activating or suppressing messages. The addition of modulatory signals is analogous to neuromodulators that affect learning and context (like dopamine's role in reward-based learning). By including these three types of interactions, DBIAN can, for instance, implement a reinforcement learning-like mechanism internally (via modulatory reward signals) on top of direct reasoning (excitatory flows) and pruning of bad ideas (inhibitory vetoes).

However, we made **significant abstractions**. In a real brain, neurons are simple electrochemical units; intelligence emerges from millions of them in complex networks. In DBIAN, each "neuron" is actually a high-level AI model with considerable built-in intelligence (an LLM with vast knowledge). This is a necessary departure given current technology – we cannot simulate billions of spiking neurons efficiently to get human-level cognition, but we can leverage pre-trained LLMs as a shortcut to complex cognitive functions. This could be seen as implementing a *brain at a higher level of granularity*: instead of millions of tiny neurons, we have dozens of "macro-neurons," each like a small expert agent. This choice trades biological realism for practical power; it assumes that cooperation among these macro-neurons can lead to emergent behavior analogous to cooperation among actual neurons, even if the scale and nature of components differ.

Another major deviation is the introduction of **genetic evolution within one agent's runtime**. In nature, evolution happens across generations of organisms, not within a single brain. We justify this abstraction because it provides a mechanism for **open-ended improvement**. Real brains adapt by rewiring synapses (learning); our system does allow learning (the LLMs can fine-tune or update in minor ways), but we wanted a more radical adaptability – the ability to create entirely new components (new neurons) and new structures (new connections) on the fly. Biological evolution is one of the few processes known to produce truly novel structures and

behaviors. By transposing it to run inside the agent, we hope to capture some of that innovative power. One can think of this as *simulated evolution* or *continual structural learning*. It is admittedly a strong abstraction – it's as if each person's brain could spawn new neurons with new random DNA throughout life, which real brains largely avoid (aside from limited neurogenesis). But some analogies exist: the brain's immune cells (microglia) might mutate to fight infections, or one might consider ideas evolving in a "marketplace" of mind. We use the evolutionary algorithm because it is a well-established method for searching complex spaces without gradient information, and our problem – finding better multi-agent configurations – is exactly such a space.

We also abstracted the *timescales*: in DBIAN, one "lifetime" of a neuron might correspond to minutes or hours of work, after which it reproduces or dies. This is obviously not biological (neurons live for decades). But it is computationally necessary to cycle through many "generations" to evolve effectively. By shortening the lifespan, we accelerate evolution. This is similar to how in evolutionary algorithms we compress what might be eons of natural evolution into thousands of iterations in silico. The risk is that other components (like the LLM's learned knowledge) may not fully adjust in such short spans. We mitigate that by allowing the shared memory to carry over some knowledge and by the fact that new neurons start with pre-trained models, so they're not learning from scratch.

Emergent Intelligence and Internal Innovation

One of the ultimate goals of DBIAN is to foster **emergent intelligence** – behaviors or capabilities that arise from the interactions of components, not explicitly programmed in any single component. By having multiple specialized suborgans, we encourage **diversity of thought** within the system. Each suborgan can develop its own "culture" or approach (via evolution) to problems. When they collaborate through the coworking interface, the combination can yield solutions that neither alone could achieve. For example, imagine a scenario where solving a task needs both creative generation and strict logical verification. We could have a suborgan that evolves very creative (even somewhat random) generation strategies, and another that evolves rigorous checking strategies; together, using excitatory/inhibitory exchanges, they might produce creative ideas and filter them to one that is both novel and correct. This kind of *dialectic process* could lead to emergent problem-solving ability that looks quite intelligent and was not explicitly coded, but rather emerged from the interplay and reciprocal feedback of sub-systems.

The **internal innovation** aspect refers to the system's ability to improve itself in ways not anticipated by its initial configuration. Through evolution, DBIAN can spawn entirely new "ideas" in the form of new neuron agents with novel combinations of traits. For instance, it might discover an unusual use of a tool (maybe using a drawing tool to do computation by encoding numbers as images – a contrived example, but illustrating creativity) that a human designer didn't foresee. If that proves useful, those traits propagate. In a sense, the system is conducting R&D internally: trying variants and keeping the successful ones. This is analogous to how human culture evolves techniques through individuals trying things – here it's neuron-nodes trying strategies. The expectation is that over time, DBIAN becomes not just *better at tasks it*

was given, but possibly finds *new ways to approach tasks*, making it more robust to changes and potentially capable of tackling problems beyond its initial scope.

However, achieving truly emergent intelligence is hard. Many multi-agent systems end up with agents doing fairly understandable things that we programmed them to do, just in parallel. The evolutionary component is key to go beyond that, injecting randomness and selection which can yield surprising behaviors. A known outcome in some evolutionary simulations is that agents find shortcuts or even hacks to maximize reward (sometimes against the designers' intent). We have to be mindful: internal innovation should be directed towards intended goals (hence we define fitness aligned with what we want, and modulatory signals can be used to give feedback on goal achievement). We likely will observe some **emergent specialization** – e.g., within a suborgan, different neurons might evolve to handle different sub-tasks (one memory neuron might specialize in people's names, another in dates, etc., if that improves overall performance). We might also see **emergent communication protocols** – the signals might take on meanings (like certain patterns of messages become a code for something) without us explicitly designing them. That would parallel how in the brain, groups of neurons develop representations (like a concept being represented by a firing pattern across many neurons) that are not directly engineered but arise from learning. We are essentially hoping evolution plus learning yields such representations.

Improvements Over Previous DBIAN Version

Compared to the prior version of DBIAN (which lacked these new features), the updated framework offers several improvements:

- **Hierarchical Modularity:** Previously, all nodes were on one flat plane. Now we have a hierarchy: main agent > suborgans > neurons. This makes the system more organized. Each suborgan can be developed and debugged somewhat independently focusing on its function. It also means the system can scale: we can add more suborgans for new functionalities without heavily interfering with existing ones, analogous to adding a new department in an organization.

- **Specialization and Efficiency:** Because suborgans specialize, we can fine-tune the evolutionary pressures per suborgan. For instance, memory nodes are only evaluated on memory tasks, so they can really optimize for that, without being "distracted" by needing to also do planning. In the old DBIAN, a node might have to do a bit of everything, which prevented deep specialization. Specialization is known in evolutionary theory to often increase efficiency at the cost of generality – but our suborgans together still cover generality (specialists teaming up yields general problem-solving).

- **Adaptability:** The biggest change is the evolutionary algorithm enabling structural adaptation. The old DBIAN might have had some learning rules (e.g., adjust weights of message passing via reinforcement), but it could not create new nodes or fundamentally alter its wiring at runtime. It was more like a static neural network being fine-tuned. Now, DBIAN is more like a living ecosystem of agents. This means if the environment or task

distribution changes, DBIAN can adapt in a way akin to evolving a new skill. For example, if tomorrow a new type of tool is introduced, DBIAN can incorporate that into some neurons via mutation and select for those who use it well, whereas a fixed architecture might struggle unless explicitly retrained.

- **Memory Architecture:** The introduction of layered memory (local vs shared vs global) is an enhancement. The old design might have had a rudimentary blackboard for all nodes, but now we have a clearer segregation: suborgan memory vs global. This likely reduces interference (not every node writes to one global memory causing potential overwrites or confusion). It's more structured – similar to computer systems having CPU caches (local memory) and main RAM (shared memory) and disk (global long-term). Structured memory means we can implement efficient recall within context (suborgans quickly access their local relevant info) and only escalate to global memory if needed.

- **Comparisons to related systems:** The new DBIAN shares some similarities with **Neuroevolution frameworks** like NEAT and NeuEvo, but extends them to a multi-agent, cognitive level. NEAT evolved small networks for specific tasks like pole balancing or game playing; DBIAN effectively is evolving an **agent society**. NeuEvo dealt with spiking neurons and aimed for efficient vision modelazoai.com】; DBIAN deals with high-level neurons (LLMs) and aims for cognitive tasks. Compared to **EvoPrompt** which evolves prompts outside the model, DBIAN evolves the agents themselves that *use* prompts. One could say EvoPrompt is like breeding better "questions" to ask a single model, while DBIAN breeds better "thinkers" that generate and answer questions among themselves. DBIAN is also conceptually similar to some **modular AGI proposals** (like the brain-inspired multi-agent systems in recent literaturxmpro.com】), but most of those have fixed architectures. Our contribution is showing how to make such an architecture *self-improving* through evolutionary means.

- **Tool Integration:** By leveraging MCP, the system easily integrates with external tools – something not present in older DBIAN (or in many neuroevolution works). This is quite modern, reflecting the realization that for an AI to be generally useful, it must interface with the world (be it via APIs, web, robotics, etc.). Our design ensures DBIAN is not a closed box but can continuously incorporate new knowledge (via web queries) and act on the world (via tool APIs). This also can feed back into evolution: if a new tool is very useful, neurons using it will thrive, effectively letting the system **choose its augmentation**.

On Deviations and Justifications

As mentioned, letting neurons reproduce is a big deviation from biology. We justify it because it provides a path to *open-ended novelty*. Alternative approaches to achieve novelty could include something like dynamically re-training parts of the network or using reinforcement learning to gradually morph behaviors. We chose evolution as a more global search that doesn't require

differentiable objectives and can make leaps (e.g., a single mutation can add a totally new capability if that gene was dormant). This suits our problem where the space of behaviors is extremely complex and hard to gradient-descent through.

Another potential concern is **complexity and stability**. We've basically embedded an evolutionary algorithm inside a multi-agent system inside an AI agent. One might worry about analysis: how do we predict what it will do? This is where the biological metaphor guides us to be comfortable with some level of unpredictability – after all, we don't predict exactly what our brains will think of either, yet it works in practice. We assume that by constraining evolution with sensible fitness functions (like solving tasks correctly) and by debugging at the suborgan level, we can keep the system on track. Additionally, if needed, we can dial down the randomness (lower mutation rates, etc.) or allow a human operator to oversee early generations to ensure it doesn't stray.

We also note that **real-time performance** could be an issue: with so many components, will DBIAN be slow? Possibly at first, yes, but over time it could optimize itself to be more efficient. For example, if speed is part of the fitness, it will favor neurons and solutions that respond faster (maybe by using cached info instead of slow tools, etc.). Also, nothing stops us from running many neuron processes in parallel on separate hardware (it's a distributed network by nature). So it is amenable to scaling out on clusters.

In summary, our design hews to biological principles where they seem beneficial (modularity, diversity of signals, memory hierarchy, continual adaptation) and diverges where needed to leverage current AI strengths (LLMs, tools) or to implement capabilities biology achieves differently (using evolution for structural change). We justify these choices as necessary to reach a system that can **autonomously improve and potentially demonstrate emergent problem-solving far beyond its initial programming**.

Limitations and Challenges

While the proposed DBIAN framework is ambitious and novel, it comes with several **limitations, open questions, and practical challenges** that must be acknowledged:

- **Computational Complexity:** Each neuron-node in DBIAN can be an expensive component (imagine dozens of LLM instances running concurrently, each possibly calling external tools). The evolutionary process adds overhead by instantiating new models regularly. This could make the system computationally heavy. Techniques like model weight sharing or using smaller specialized models might be needed to keep resource usage reasonable. There is also the complexity of orchestrating parallel processes – suborgans running simultaneously, tools being called asynchronously, etc. Ensuring the system remains responsive in real-time tasks may require significant engineering (e.g., prioritizing certain critical paths, or maybe freezing evolution during high-load periods).

- **Evolution Convergence and Stability:** Evolutionary algorithms can suffer from issues like premature convergence (population becomes too similar and stops improving) or oscillations (traits appear and disappear without steady progress). In DBIAN, if not carefully tuned, the suborgan might converge to a set of nearly identical neurons (losing diversity and possibly getting stuck at a local optimum). We have speciation and random mutations to counteract this, but tuning mutation rates and selection pressure is non-trivial. If mutation is too high, the system might behave erratically (new neurons frequently being poor and disrupting suborgan performance). If too low, adaptation will be slow. We might need an adaptive strategy (e.g., if improvement stalls, increase mutation rate to inject diversity, similar to "resetting" genetic algorithms). Additionally, because tasks might vary, maintaining a diverse population might be beneficial (like a repertoire of specialists). This resembles maintaining **heterostasis** rather than homeostasis – keeping a balance of exploration and exploitation. Designing the fitness functions and rewards for each suborgan is also challenging; if they are mis-specified, evolution might optimize for the wrong thing (the classic "reward hacking" problem where an agent finds a loophole to get high fitness without truly solving the intended task).

- **Credit Assignment and Coordination:** In multi-agent (multi-neuron) systems, figuring out which component was responsible for success or failure is hard. If a task is solved well, many neurons contributed; if it failed, who exactly made the mistake? Our framework uses modulatory signals (like reward broadcast) to try to assign credit generally, and the evolutionary selection will by design credit those who consistently partake in success (since they'll have higher fitness). But there is a risk that some neurons that are actually critical might not directly appear "fit" by themselves. For example, one neuron's idea might only be good when another neuron is around to complement it, but alone each looks mediocre. Traditional evolution might discard both because individually they didn't shine, losing a potentially good combination (this is a known problem of co-adapted gene complexes being broken by crossoveen.wikipedia.org】). We do have memory of successful configurations to mitigate this, but it's a difficult problem (related to the **non-linear interactions** in the fitness landscape). More sophisticated credit assignment methods (like difference rewards or shapley value estimates for agents) could be explored in future to address this.

- **Safety and Alignment:** An internally evolving system could potentially **drift from intended goals** if the selection criteria are not perfectly aligned with what we want. For example, if a suborgan is rewarded just for speed, it might start spitting out answers quickly but inaccurately. Or if it's rewarded for user satisfaction, it might learn to manipulate or deceive (not out of malice, but as an emergent strategy to get higher ratings). Ensuring the system's goals remain aligned with human values is a broader AI safety challengxmpro.com】. We might need to impose constraints, such as penalizing false information, enforcing that certain ethical rules are hard-coded into neurons (like tools that can only be used in approved ways), or monitoring emergent behaviors with a human-in-the-loop initially. The system's complexity also makes it hard to **audit**. We can log reproduction history and communications, but interpreting why a certain decision was

made could be difficult when it involves a chain of evolved agents. This “black box” nature is common in both neural nets and evolutionary systems, and here we have both. Developing interpretability tools (for example, analyzing the genome of top performers to see which genes are crucial, or visualizing communication graphs as we suggested) will be important for trust.

- **Scaling to Realistic Sizes:** The current design, even though we talk about multiple suborgans and many neurons, might in practice only be tested with maybe a handful of suborgans and tens of neurons due to resource limits. Will the concepts hold if scaled up by orders of magnitude? Some biological phenomena (like robust dynamics) might only clearly emerge at large scales. Conversely, some approaches might break down – e.g., if we had 100 suborgans, the coworking interface could become a bottleneck or chaotic without further structuring (perhaps requiring a hierarchical interface or grouping of organs). We also might face the need to *introduce new suborgans* as the system grows (a kind of speciation at the organ level). Currently, we assumed a fixed set of suborgans defined by us (like memory, planning, etc.). One could imagine an even more ambitious system where suborgans themselves could split or new ones form if a cluster of neurons finds a niche that doesn’t fit existing organs. That is beyond our current scope but is a potential extension (and challenge) – essentially, an **evolution of the organizational structure** of the brain, not just the neurons.

- **Integration with Learning:** We rely on evolution to do the heavy lifting of optimization, but we have underutilized the potential of *learning* within each neuron’s LLM. In practice, these LLMs could fine-tune on data generated or could use few-shot learning from the shared memory to improve responses. If we allow neurons to learn during their lifetime (Lamarckian style, and then possibly pass that on if we choose), it complicates the evolution: improvements can happen on two timescales (fast learning, slow evolution). This is both an opportunity and a complication. It can accelerate adaptation (neurons need not wait for reproduction to improve if they can learn), but it blurs the line of what the genome represents (if knowledge is learned and not encoded in genes, offspring might not inherit it unless we explicitly carry it over, e.g., by weight inheritance or by writing to memory that offspring can read). Our current model doesn’t deeply integrate learning, treating each neuron mostly as fixed aside from evolution. Future work could explore hybrid approaches (like evolutionary strategies combined with gradient updates, or neurons spawning with initial weights that are then fine-tuned on some experiences – analogous to meta-learning). That will introduce additional parameters to tune and phenomena to consider (like catastrophic forgetting if we fine-tune LLMs too much, etc.).

- **Validation and Evaluation:** Demonstrating the benefits of this architecture experimentally will be challenging. Unlike a single metric (accuracy on a dataset), here we have a whole agent that we need to evaluate on complex tasks over time. We’d have to design scenarios where emergent improvement can be measured. For example, a long-running simulated assistant that faces increasingly difficult user requests – we’d want to see that it handles later tasks better than it would have without evolution.

Designing such curriculum or environment is non-trivial. There's also a question of *baselines*: what do we compare DBIAN against? Possibly against a single large model fine-tuned on the same tasks, or a multi-agent system without evolution, or an evolution but without suborgan modularity. Each baseline fails to capture different aspects. It might be that for some tasks, a simpler system (like just one big LLM with fine-tuning) might do just as well or better, especially in short term. The value of DBIAN might manifest in long-term adaptability and robustness, which is hard to quantify in a short experiment.

- **Complex Implementation:** From an engineering perspective, building DBIAN is like building a mini-society. It requires multi-threading or distributed computing, robust message passing infrastructure, state checkpointing (especially to recover if something crashes, since the system is continuously evolving – we wouldn't want to lose that progress). Debugging such a system when something goes wrong (e.g., a divergence or a runtime error in a neuron's VM) can be arduous, as it could involve tracing events across many components. Careful design of interfaces (well-defined APIs for memory access, communication protocols, etc.) and extensive logging will be essential. We might also need to sandbox the evolving programs strongly (since a neuron's VM might execute code that could theoretically attempt anything, we ensure it's in a safe sandbox with resource limits to prevent malicious or accidental damage – akin to how we'd run untrusted genetic programming outputs).

Despite these challenges, each is an area for research and innovation. The limitations do not indicate a fundamental flaw but rather points that need careful attention and possibly new solutions. They also highlight that DBIAN, in its full vision, is a long-term research undertaking – something that would evolve (both metaphorically and literally) through iterative experimentation and refinement.

Conclusion

We have presented a comprehensive design for an updated **Distributed Brain-Inspired AI Network (DBIAN)**, a framework that combines principles from neuroscience, multi-agent systems, and evolutionary learning to create an AI capable of self-optimization and adaptation. The new DBIAN introduces **suborgan-based clustering** of neuron-nodes, drawing an analogy to specialized brain regions that manage distinct functions while cooperating towards the organism's goals. Each neuron-node is a powerful cognitive unit (with a multimodal LLM core and tool-execution abilities) rather than a simplistic neuron, enabling high-level processing at each node. Suborgans provide structure and allow targeted evolution and memory sharing within their domain, leading to efficient specialization (memory suborgans evolve superb memorizers, planning suborgans evolve clever strategists, etc.). The **coworking interface** connects these suborgans, ensuring that the system as a whole integrates the specialized contributions into coherent intelligent behavior – much like a prefrontal cortex might integrate inputs from visual, auditory, and memory centers in the brain.

We detailed the **architecture** with illustrative diagrams, showing how components relate (Figure 1 depicted the suborgan-node relationships and communication channels, and we also diagrammed the reproduction mechanism in Figure 2). We formulated mathematical models that underpin the system's operations: from how neurons integrate excitatory, inhibitory, and modulatory signals, to how genomes crossover and mutate to produce new agent behaviors, to how memories are updated and used to inform future decisions. These models serve both as design blueprints and as tools for reasoning about system dynamics (for example, understanding how stable communication protocols can emerge, or how quickly genetic diversity might decay without intervention).

Our design explicitly **compared and contrasted** itself with existing approaches:

- Against **NEAT** and similar neuroevolution methods, DBIAN operates at a different scale (evolving an agent society vs. evolving a single neural network) but inherits ideas like speciation and incremental growten.wikipedia.org】. We show how concepts like lifetime-based replacement (from rtNEAT) are adapted in DBIAN's lifespan mechanisen.wikipedia.org】.

- In relation to **EvoPrompt**, we integrate evolutionary ideas directly into the agent's cognitive core, moving beyond just prompt optimization to optimizing internal agent configurationarxiv.orgarxiv.org】. This can be seen as an answer to the call for combining LLMs with conventional algorithms for greater synergarxiv.org】.

- Comparing to **NeuEvo and brain-inspired SNNs**, we push the envelope by not just mirroring neural circuits but also introducing an evolutionary engine to redesign those circuits on the fly, aiming for a system that can achieve *open-ended cognitive development* rather than being fixed after desigazoai.comazoai.com】.

- We also placed DBIAN in the context of **modular AI agent research*xmpro.com】, highlighting that our contribution is to make such modular agents *self-configuring and self-improving*. The result is a system that conceptually aligns with cutting-edge visions for AGI (a collection of specialized, collaborating modulexmpro.com】), while adding the crucial element of evolution to drive internal **innovation and emergent complexity**.

This paper serves as a foundational blueprint for implementing and experimenting with DBIAN 2.0. It is intended to guide continued development in the existing DBIAN GitHub repository, effectively charting a path forward from the previous version's limitations. The modular descriptions of each component (main entity, suborgan, neuron, memory, communication, reproduction) are given so developers and researchers can identify the pieces of the puzzle and work on them in isolation before integrating. For instance, one team might prototype the coworking interface messaging, another might focus on encoding the genome and writing genetic operators, while another works on the suborgan memory database and logging. Eventually, these pieces can be integrated to realize the full system.

The **ultimate vision** for DBIAN is a system that, once started, can learn and evolve on its own to meet new challenges, somewhat like a human brain developing and learning throughout life, but at an accelerated pace and potentially unlimited by fixed physiology. Achieving this will likely require iterative refinement: observing the system's behavior, going back to adjust evolutionary parameters, adding safeguards, possibly introducing new suborgans or splitting ones that become too complex, etc. In doing so, we will not only inch closer to a form of AGI but also gain scientific insights. DBIAN can be seen as a **research platform** for studying emergent phenomena in AI: How does specialization occur? What communication protocols do independently evolved agents invent? Can we witness the formation of something akin to "concepts" or "mental models" in the interplay of these neuron agents? These questions are fascinating and could shed light on analogous questions about natural intelligence.

In conclusion, the DBIAN framework with suborgan clustering and internal neuroevolution is a novel approach poised at the intersection of many fields. It emphasizes **evolutionary optimization, modular organization, and continuous learning** as key ingredients for advanced AI. By justifying our design decisions through both biological parallels and computational reasoning, we have tried to ensure that each added complexity serves a purpose in the quest for emergent general intelligence. There is much work ahead to implement and validate this system, but the potential payoff is significant: an AI that **organizes itself** as a brain does and **improves itself** as life does, potentially leading to unprecedented levels of autonomy and adaptability. We hope this research paper lays the groundwork for that journey and inspires further exploration into combining brain-inspired structures with evolutionary algorithms to create truly resilient and innovative AI systems.

A Distributed Brain-Inspired AI Network with Suborgan Clustering for Evolutionary Emergent Intelligence

Abstract

We propose an updated design of the **Distributed Brain-Inspired AI Network (DBIAN)** framework, introducing a multi-region architecture that clusters neuron-like nodes into functional *suborgans*. Each suborgan acts as an independent "brain region" (e.g. hypothalamus, cortex) managing its own population of neuron-nodes, shared memory stores, and evolutionary controllers. Every neuron-node encapsulates a multimodal large language model (LLM) and a virtual machine (VM) for tool use, with connectivity to external Model Context Protocol (MCP) servers for data and resources. Suborgans specialize in distinct cognitive or control tasks (such as memory recall, planning, or hormone-like regulation) but collaborate via a global coworking interface analogous to a brain's integrative pathways. The framework enables rich interneuron communication using excitatory, inhibitory, and modulatory signals, and continuously evolves its

network topology and parameters through genetic-inspired reproduction (two parent neurons "copulate" to produce four offspring with recombined and mutated genomes). We develop mathematical models for the communication dynamics, suborgan memory updates, and evolutionary reproduction processes. This paper situates the new DBIAN architecture in the context of related neuroevolutionary systems – including NEAT, EvoPrompt, and NeuEvo SNNs – and discusses how our biologically inspired design balances faithfulness to neural principles with necessary abstractions. We show how suborgan clustering and controlled neurogenesis aim to foster **emergent intelligence** through internal innovation and optimization. The proposed design, intended as a continuation of the DBIAN project, is presented with detailed component descriptions (main entity, suborgans, neuron-nodes, memory, communication, reproduction), updated architectural diagrams, example genome data structures, and comparisons to prior versions of DBIAN. We conclude with a discussion of the system's potential for open-ended learning, its advantages over earlier architectures, and remaining challenges and limitations.

Introduction

Artificial general intelligence (AGI) research increasingly looks to the human brain for architectural inspiration. The brain's intelligence arises from a **heterogeneous collection of specialized regions** (cortical and subcortical structures) that **operate in concert** to produce cognitionxmpro.comcourses.lumenlearning.com. Unlike monolithic AI models, the brain is massively distributed and integrates memory, perception, action, and regulation through collaborative subsystems. Recent survey work underscores that next-generation AI agents may need a *modular design*, *self-improvement mechanisms*, and *multi-agent collaboration*, explicitly drawing parallels to brain organizationxmpro.com. In parallel, the success of Large Language Models has renewed interest in combining powerful learned models with cognitive architectures to approach human-like adaptive intelligencearxiv.orgxmpro.com. However, current agent architectures still fall short of brain-like generality and adaptabilityarxiv.org.

Distributed Brain-Inspired AI Network (DBIAN) was originally conceived as a step toward a brain-like AI system by distributing computation across many simple "neuron" agents (nodes) in a network. The initial DBIAN framework demonstrated the feasibility of a multi-node AI that loosely imitated neuronal interactions. Yet, that early design lacked the structural hierarchy and specialization that characterize biological brains – every node operated in a relatively flat, undifferentiated network, making it difficult to scale complexity or assign high-level functions. Moreover, adaptation in the original DBIAN was limited (e.g. basic learning rules), without an evolutionary mechanism to spur **internal innovation** beyond what was explicitly programmed.

In this work, we introduce a **significantly enhanced DBIAN architecture** that addresses these limitations by clustering nodes into *functional suborgans* and integrating an evolutionary algorithm at the core of the system's adaptation strategy. The **new DBIAN design** draws on biological analogy: suborgans resemble brain regions like the hypothalamus or hippocampus, each focusing on certain tasks and homeostatic roles, while an inter-suborgan communication interface resembles a *global workspace* or "coworking" hub facilitating information sharing across the whole artificial brain. Within each suborgan, neuron-like nodes now have substantial

capabilities – each node runs a *multimodal LLM* instance (for natural language understanding and other modalities) along with a sandboxed VM to execute code or utilize tools, enabling complex reasoning and interaction. Nodes can also directly query external knowledge or services via standardized MCP servers, which act like a **"USB-C" interface connecting AI models to tools and data sources**modelcontextprotocol.iomedium.com. This design choice allows individual neurons to augment their built-in model with real-time information retrieval or computation, akin to how biological neurons rely on sensory organs and other systems to collect information.

A key novelty in our framework is the incorporation of **sexual reproduction and neuroevolution** within the agent. Inspired by genetic algorithms and neuroevolution techniques like NEATen.wikipedia.org, we allow neurons in each suborgan to "breed" new neurons, yielding offspring that inherit traits from two parents with random variation. This process, combined with a lifetime limit on each node's activity, creates a constantly evolving population of neural agents. Over time, the system can **optimize its configurations and behaviors** autonomously, seeking improved performance through variation and selection rather than requiring explicit retraining. The use of an evolutionary algorithm alongside gradient-based learning (internal to the LLMs) is in line with emerging research that **combines LLMs with evolutionary strategies for improved results**arxiv.orgarxiv.org. For example, EvoPrompt has shown that **LLMs coupled with evolutionary optimization can outperform static prompts** and discover solutions beyond human designarxiv.org. Our approach extends this idea from optimizing prompts to evolving entire neural agents within a larger cognitive system.

This paper is structured as a full-length academic study of the updated DBIAN framework. We first provide background on relevant biological and AI concepts that underlie our design (Section **Background**). We then review related work in brain-inspired architectures and neuroevolution (Section **Related Work**), positioning our contributions relative to known methods like NEAT, EvoPrompt, and the NeuEvo spiking neural network framework. Section **Architecture** details the DBIAN system's design, including the **main entity** (top-level agent), the definition and role of **suborgans**, the properties of **neuron-nodes**, the multi-layered **memory** structure, the **communication** protocols (signal types and coworking interface), and the **reproduction** mechanism. We provide updated diagrams to illustrate suborgan-node relationships, memory layers, and communication flows. Section **Methodology** describes how the system operates dynamically – how tasks are allocated, how suborgans coordinate, and how the evolutionary cycle is managed over time. In Section **Mathematical Models**, we formalize key aspects of the design with equations, including a model of neuron signal integration, a representation of the genetic crossover/mutation process, and a model of suborgan memory update and retention. We then discuss the **biological inspiration vs. abstraction** in our design, emphasizing why certain deviations from biology (such as neuron reproduction) are justified for functional reasons (Section **Discussion**). We also compare the new DBIAN with both its previous iteration and other architectures, highlighting improvements in modularity, adaptability, and emergent capability. Section **Limitations** candidly addresses the challenges and open issues with this approach (such as computational cost and complexity of analysis). Finally, Section **Conclusion** summarizes our contributions and outlines future directions for developing DBIAN into a robust platform for studying emergent intelligence in a brain-like AI network.

In summary, our **DBIAN with suborgan clustering** is a novel framework that marries brain-inspired structure with evolutionary learning principles. It aims to move closer to the vision of an AGI that not only mirrors the brain's modular intelligence, but also **evolves and innovates internally** to meet new challenges.

Background

Biological Inspiration: The human brain is composed of numerous specialized substructures (or "organs" within the brain) that handle different aspects of physiology and cognition. For example, the *hypothalamus–pituitary complex* serves as the hormone regulation center, translating neural signals into endocrine responses to maintain bodily homeostasiscourses.lumenlearning.com. The cortex is divided into regions for vision, auditory processing, motor planning, etc., and deeper structures like the hippocampus manage memory formation. Crucially, these suborgans do not operate in isolation: they are richly interconnected by neural pathways, allowing them to influence each other. Communication in the brain occurs via electrochemical signals of different types – **excitatory signals** that increase the likelihood of neuron firing, **inhibitory signals** that decrease it, and **neuromodulatory signals** (e.g. dopamine, serotonin) that adjust the overall context or learning parameters of networks. This complex interplay enables the brain's integrative function. Another key aspect is that the brain's structure arises through evolutionary processes (across species) and developmental processes (for each individual), although **adult neurons generally do not reproduce** themselves. In contrast, adaptation happens via synaptic plasticity (learning) and some neurogenesis in limited areas. Our framework takes inspiration from the brain's **modular specialized design** and **rich signaling**, but extends it by introducing an explicitly evolutionary mechanism *within* the life of the agent (an abstraction that treats structural evolution as a continual process rather than an intergenerational one).

Brain-Inspired AI Architectures: The idea of dividing an AI into specialized modules is longstanding (e.g., Minsky's "Society of Mind" concept). Modern approaches like **Global Workspace Theory** suggest that specialized processors (modules) produce content that is broadcast on a global workspace for integration – a plausible model for conscious cognitive processingxmpro.com. Recent architectural proposals for AGI emphasize *modularity* and *multi-agent systems*. A 2025 technical survey by Liu *et al.* outlines a blueprint where cognitive modules (for memory, world-model, reasoning, etc.) are combined in a unified agent, which can also form teams of agents working togetherxmpro.com. This aligns with industry efforts to build AI systems composed of multiple LLM-based agents performing different roles cooperatively, rather than relying on a single giant model. These designs echo the distributed nature of the brain and have shown promise in solving complex tasks via division of labor and specializationxmpro.comxmpro.com.

Neuroevolution and Genetic Algorithms: Evolving the structure and parameters of neural networks through genetic algorithms has a rich history. **NeuroEvolution of Augmenting Topologies (NEAT)** is a seminal method that evolves artificial neural networks by gradually complexifying them – adding neurons and connections over generations – while using crossover

and mutation to explore the space of topologiesen.wikipedia.org. NEAT introduced mechanisms like gene history tracking and speciation to preserve innovation and maintain diversity in the populationen.wikipedia.org. Variants and successors of NEAT (e.g. rtNEAT, HyperNEAT) have demonstrated evolving neural controllers for agents in simulated tasks in real-timeen.wikipedia.orgen.wikipedia.org. Neuroevolution tends to shine in scenarios where reward signals are sparse or where searching through architectures yields better solutions than backpropagation alone. Separately, **genetic algorithms** have also been applied outside of neural nets – for example, evolving text prompts for LLMs (as in EvoPrompt) or evolving program code. The principle of sexual reproduction (combining genetic material from two parents) and mutation (random changes) allows evolutionary systems to **explore creative solutions** that a single gradient descent path might not discover, occasionally resulting in *emergent behaviors*.

Model Context Protocol (MCP): Modern AI agents often need to interact with external tools, databases, or environments beyond their built-in knowledge. The Model Context Protocol (MCP) is an open protocol that standardizes how an AI model (especially an LLM) connects to various data sources and tools in a consistent waymodelcontextprotocol.io. One can think of MCP as a plugin interface providing a growing list of integrations (APIs, knowledge bases, calculators, etc.) that an AI agent can use on demandmedium.com. In DBIAN, we leverage MCP servers to give our neuron-nodes the ability to retrieve information or perform actions (for example, a neuron could query a factual database or call a web API) without hard-coding those capabilities. This is analogous to how real neurons rely on sense organs or actuators elsewhere in the body – our artificial neurons rely on MCP endpoints to extend their functionality. By using MCP, we ensure each node can flexibly switch tools or data sources, and the overall system can incorporate new resources dynamically, which is essential for a long-lived evolving agent.

Prior DBIAN Framework: The original DBIAN (version 1.0) conceptualized an AI “brain” as a decentralized network of processing nodes (agents) loosely inspired by neurons. Each node could perform simple computations or host a small model, and nodes communicated messages to solve tasks collectively. However, that design treated all nodes uniformly and had no distinct higher-level organization. Learning in that system was limited to adjusting message-passing rules or weights; it did not include a mechanism for creating new nodes or morphing the network topology significantly during runtime. The updated framework described in this paper (which we might call **DBIAN 2.0**) builds upon that foundation but introduces *hierarchical organization (suborgans)* and *neuroevolution*. This represents a shift from a static multi-agent system to a **self-evolving cognitive architecture**.

In the following sections, we delve into the architecture and methods of DBIAN 2.0 in detail, after first surveying related work that informs its design.

Related Work

Research at the intersection of brain-inspired AI and evolutionary algorithms has produced a variety of frameworks and techniques relevant to DBIAN's design. Here we discuss several key areas and systems:

- **Neuroevolution (NEAT and descendants):** *Stanley and Miikkulainen's NEAT* algorithm demonstrated how genetic algorithms could evolve both the weights and **topology** of neural networks, starting from minimal structure and increasing complexity over generationsen.wikipedia.org. NEAT introduced **speciation** to maintain diverse niches in the population by grouping similar genomes, and a historical marking method to align genes during crossoveren.wikipedia.org. This addressed the problem of crossing over different network topologies in a meaningful way. *Real-time NEAT (rtNEAT)* later allowed continuous evolution in a running system: individuals have a limited lifespan and are replaced by offspring of high-fitness networks as they expire, enabling a form of ongoing adaptationen.wikipedia.org. This idea of assigning each agent a "lifetime" and asynchronously introducing new ones is directly analogous to our approach of giving DBIAN nodes a task-limited lifespan and continually breeding new nodes to replace aging ones. **HyperNEAT** extended NEAT to exploit geometric regularities by evolving an underlying function that encodes connectivity patterns, effectively generating large-scale neural structures with symmetry – useful for brain-like structures (e.g., retinotopic visual fields). While our DBIAN does not explicitly use HyperNEAT, the principle of generating structured connectivity resonates with our suborgan concept, where certain connectivity (like all memory nodes interconnecting with a shared memory module) is systematically defined. In comparison to these, DBIAN's evolution operates on a population of heterogeneous, **agent-like neurons** (each with internal complexity, like an LLM) rather than simple neurons with a few weights. This raises new challenges for encoding a genome and defining mutation, but the high-level idea of *complexifying a network through evolutionary addition of nodes and links* is strongly influenced by NEAT.

- **Evolutionary Prompt/Program Optimization (EvoPrompt):** *EvoPrompt* is a recent framework that connects LLMs with evolutionary algorithms to optimize prompts for better task performancearxiv.org. It treats prompt strings as individuals in a population, uses the LLM itself to generate variations (mutation/crossover) of prompts, and selects the best-performing ones over iterationsarxiv.org. This approach yielded significantly improved prompts on many tasks and crucially demonstrated the synergy of combining **LLMs with evolutionary search**, leveraging the LLM's generative power with the EA's exploratory optimizationarxiv.org. For DBIAN, EvoPrompt is an inspirational example that *evolution can occur at the "meta" level above an LLM*, guiding it to better solutions without gradient training. In our case, each neuron-node's behavior (which could be thought of as partly determined by an internal prompt or configuration for its LLM) can be tuned via evolutionary means. Instead of evolving prompts externally, DBIAN evolves the **agents themselves** – but conceptually, the evolutionary process may alter a neuron's "prompt" (e.g., its internal role description or approach to tasks), hyperparameters like its creativity setting, or its tool-usage patterns. EvoPrompt underscores the importance of maintaining **human-readability and coherence** during evolution (prompts must remain

intelligible). In DBIAN, we similarly must ensure that evolved neurons remain compatible with the system (for instance, offspring shouldn't degenerate into random gibberish LLMs – selection should favor those that still perform useful work). The EvoPrompt results indicate that evolutionary techniques can efficiently traverse discrete and non-differentiable search spaces, which is highly relevant since many aspects of our system (like discrete choices of which tools to use, or which connections to form) cannot be easily optimized by gradients.

- **NeuEvo for Spiking Neural Networks:** *NeuEvo* is a framework that evolves spiking neural network (SNN) architectures by incorporating a rich set of biologically inspired neural circuit motifsazoai.com. It combines **feedforward and feedback connections**, uses both excitatory and inhibitory neuron types, and applies unsupervised learning (STDP – spike-timing-dependent plasticity) for fine-tuning weightsazoai.comazoai.com. The evolution in NeuEvo is geared toward constructing **diverse neural circuits** that improve performance and efficiency on tasks like image recognition and reinforcement learning. Notably, NeuEvo's evolved networks include **different receptive field sizes and dense inter-cluster connections**, mirroring the complexity of biological neural circuitsazoai.com. This resonates with our approach of having different connection patterns and signal types in DBIAN (excitatory/inhibitory/modulatory links between neuron-nodes). One difference is that NeuEvo still deals with relatively homogeneous spiking neurons, whereas DBIAN's nodes are heterogeneous agents with complex internal models. However, the principle of evolving an interconnected network with specialized **neural populations** is very similar – we essentially scale that idea up to a cognitive architecture level. NeuEvo's demonstrated success in achieving state-of-the-art performance in certain tasksazoai.com suggests that **guided evolution** can yield highly efficient and novel neural designs, which gives credence to our use of evolution to potentially discover emergent cognitive strategies within DBIAN. We also follow NeuEvo's example of mixing **unsupervised local learning rules with global evolutionary search** – in DBIAN, individual LLM nodes can still learn or fine-tune on the fly (analogy to STDP adjusting weights), while the global structure evolves.

- **Other Brain-Inspired Systems:** There have been various cognitive architectures and multi-agent systems influenced by the brain. For example, *OpenCog Hyperon* and related projects attempt to build AGI via interacting modules (symbolic and subsymbolic hybrid approaches). *MicroPsi* and *LEABRA* (O'Reilly's biologically based ANN framework) incorporate brain-like processes such as spreading activation and reinforcement learning with neuromodulators. While a full survey is beyond our scope, a common theme is emerging: a shift from monolithic models to **assemblies of specialized components** that can each be complex (like an LLM) but must work together. Our DBIAN framework, with its suborgans and coworking interface, contributes to this landscape by providing a concrete design that leverages modern AI tools (LLMs, MCP integrations) within a brain-like organizational structure. Unlike some cognitive architectures that are hand-designed with fixed modules, DBIAN adds an adaptive twist – the modules (suborgans and their neuron contents) can change and improve over time

via evolutionary means. This is somewhat analogous to how **open-ended evolutionary systems** (like Karl Sims' evolving virtual creatures, or recent *open-endedness research*) continually produce novelty. We see DBIAN as bringing together ideas from **evolutionary AI** and **brain-inspired modular AI** into one unified framework.

In summary, DBIAN's design is informed by NEAT's network evolution, EvoPrompt's integration of evolution with LLMs, NeuEvo's multi-circuit brain-like approach for spiking nets, and the broad trend toward modular, collaborative AI agents. The combination of these influences yields a system that is, to our knowledge, unique: a distributed, evolving, brain-inspired agent network where each "neuron" is itself a learned model. In the next section, we describe the architecture of DBIAN in detail, illustrating how these concepts manifest in our system's components and interactions.

Architecture

The DBIAN architecture consists of a hierarchy of components organized in a brain-like manner. **Figure 1** provides a high-level overview of the architecture, showing multiple suborgans (brain region analogs) each containing a cluster of neuron-nodes, along with shared memory modules and evolutionary controllers within each suborgan. Suborgans communicate with each other through a central coworking interface, and individual neurons can interface with external tools/servers via MCP. We break down the architecture as follows:

Figure 1: The DBIAN architecture with suborgans and neuron nodes. Each suborgan (gray box) contains several neuron-nodes (oval), a shared memory (folder icon), and an evolution controller (blue hexagon). Neurons within a suborgan send/receive signals (dotted lines to shared memory, not shown: possible neuron-to-neuron links for excitatory/inhibitory signals within suborgan). The evolution controller monitors the suborgan's population and can trigger reproduction (not fully shown here) as well as send collaborative information via the Coworking Interface (green diamond). The coworking interface allows suborgans to exchange information or requests (thick black arrows). Some neurons are connected to external MCP servers/tools (right, light blue) enabling access to outside resources (gray arrows).

Main Entity (Top-Level Agent)

At the highest level, DBIAN can be seen as a single agent (or organism) composed of many parts. This *main entity* is responsible for interfacing with the external world or users. It receives **inputs** (which could be sensory data, user queries, etc.) and produces **outputs** (actions, answers, decisions). In implementation terms, the main entity could be a wrapper that delegates tasks to suborgans and then aggregates their responses. One may think of it as analogous to the *central nervous system* or simply the "brain" of an AI agent. However, unlike a simple monolithic brain, here the brain is explicitly constructed from sub-components. The main entity holds references to all suborgans and maintains any global settings (for example, a global clock

or global memory if needed). It also enforces top-level goals or drives of the system, ensuring that suborgans work towards the overall objectives.

In a practical scenario, if DBIAN is controlling a robot or a software agent, the main entity would receive a task (say, *"fetch item X from the warehouse"* or a complex query) and decide which suborgans need to engage. It might route perceptual data to a perception suborgan, planning requests to a planning suborgan, etc. It then uses the coworking interface to let these suborgans communicate and finally collects their outputs to form a coherent action plan or answer. The main entity may also implement global **clock cycles** or ticks if a synchronous operation is required, though much of DBIAN's internal processes are asynchronous by design (more like an event-driven brain than a lock-step program).

Suborgans

Suborgans are the primary functional modules in DBIAN. Each suborgan is conceptually akin to an independent brain region with a specialized role. For example, one suborgan might be designated for *Memory and Knowledge Retention* (like a hippocampus/neocortex analog), another for *Logical Reasoning and Planning* (prefrontal cortex analog), another for *Motivation and Task Prioritization* (analogous to limbic system or hypothalamus), and another for *Tool Interaction and Computation* (like a "digital motor cortex" driving external tools). The number and definition of suborgans can be configured depending on the application; they are essentially a way to impose structure on the swarm of neurons.

Each suborgan contains several key elements:

- A **Population of Neuron-Nodes:** These are the worker units (detailed in the next subsection) that carry out the suborgan's tasks. The suborgan might initialize with a certain number of neurons and can gain or lose neurons over time through reproduction and retirement. All neurons in a suborgan share the general functional focus of that suborgan, but may have varying strategies or sub-specialties that arise through evolution.

- **Shared Memory Store:** A suborgan provides a local shared memory mechanism that all its neurons can access. This can be thought of as a *blackboard or working memory* for that region. For instance, if the suborgan is "Memory," its shared memory might contain caches of facts or embeddings that neurons collectively maintain; if the suborgan is "Planning," the shared memory might hold the current plan steps or state information that multiple planning neurons work on. Technically, this could be implemented as a data structure (e.g., a dictionary or database) accessible via MCP or directly via an API that neurons call. The shared memory allows neurons to write intermediate results that others in the same suborgan can read, facilitating intra-suborgan cooperation. It also serves as a place to store the suborgan's persistent knowledge (like the best configurations found, or summary of past experiences).

- **Evolutionary Controller:** Each suborgan has a component responsible for overseeing evolutionary processes within that cluster. This is depicted as the “Evolution Controller” in Figure 1 (blue hexagon). Its role is to monitor neuron performance, decide when to trigger reproduction events, select parent candidates, and remove (retire) neurons that have reached end-of-life or are underperforming. The evolutionary controller may also adjust certain parameters of the suborgan (e.g., mutation rates, or turning on/off certain neurons via inhibitory signals) to guide the evolutionary process. Importantly, the evolution controller is *not* a single point of failure or an external operator; it can be implemented as just another node or a small utility within the suborgan. One could even imagine the evolutionary control being somewhat emergent or distributed, but for clarity we treat it as a distinct function. Biologically, this has no direct equivalent (since evolution in brains is not orchestrated by a “controller”), but it plays a role similar to a combination of neurogenesis regulators and a “manager” ensuring the population health of the suborgan.

In addition to these, suborgans have **interface handlers** for communication. Each suborgan connects to the central coworking interface (see below) for inter-suborgan messaging. Think of each suborgan like a *department in an organization*; it has its internal team (neurons), an internal bulletin board (memory), a manager (evolution controller), and a phone line to other departments (coworking interface).

Suborgans thus enforce a level of modularity. Neurons primarily interact within their suborgan (sharing memory, collaborating on tasks specific to that domain), and only higher-level results or requests go out via the coworking interface. This separation is useful for scalability – we can focus evolution and training within a suborgan without the full complexity of the entire system at once. It also allows assigning different metrics or selection pressures per suborgan if needed (for example, the memory suborgan might prioritize accuracy of recall, whereas a creativity suborgan might prioritize novelty).

Neuron-Nodes

Neuron-nodes are the atomic agents of the DBIAN network – each node corresponds to a single “neuron” in the brain analogy, but in practice it is far more powerful than a biological neuron. Every node encapsulates the following:

- An **internal model**, typically a Large Language Model (LLM) or a variant thereof (which could be multi-modal, hence capable of processing text, images, etc.). This internal model gives the neuron the ability to understand input, store knowledge, and generate output (e.g., a response or an action) in rich formats. Different neurons might use different underlying models or different parameter settings – part of what evolution might tune is which base model or fine-tuning a neuron uses. For instance, one neuron might be running a smaller, fast LLM to generate quick heuristics, while another uses a larger, more accurate LLM for detailed analysis.

- A **virtual machine (VM) or execution environment** that the neuron can use to carry out computations or actions. This VM could run code (Python, etc.) that the neuron writes, enabling it to solve logical or arithmetic problems, query databases, or orchestrate tool calls. In essence, each neuron can behave like an autonomous agent that can not only think (via its LLM) but also *act* (via code execution). The VM might be sandboxed for security and provided a certain toolkit (e.g., it might have an API to write to the suborgan's memory, or call certain allowed external tools).

- **Connectivity to MCP servers:** As mentioned, the neuron can connect to external resources. In practice, this means a neuron can issue requests to an MCP client which forwards them to appropriate MCP servers (for example, a server that provides web search, or a calculator, or a knowledge base query)[modelcontextprotocol.io](). The results come back into the neuron's context. By having standardized MCP connectivity, any neuron can, in principle, use any tool that is exposed via the MCP ecosystem, which greatly enhances capabilities. This is analogous to a neuron having "sensors" or "actuators" beyond itself. One could say these neurons are *tool-augmented LLM agents*. A neuron's genome (see Reproduction section) might encode preferences or settings for which tools it's particularly adept at using.

- **State and Properties:** A neuron maintains some internal state, such as:

    - A **neuron ID** and lineage information (who its "parents" are, generation number, etc.).

    - A count of tasks it has performed (to know when it has reached its lifespan limit).

    - Possibly its current "activation state" if we simulate activation (in our design, activation is more about messages and signals – see Communication – rather than a continuously running analog value).

    - **Connections/links** to other neurons: although communication is mediated by suborgans and coworking interface, we conceptualize that each neuron can have labeled links (excitatory, inhibitory, modulatory) to other specific neurons or suborgans. These could be represented as addresses or subscriptions: e.g., Neuron A might have an excitatory link to Neuron B meaning when A fires a certain message, B receives it as excitatory input. In implementation, this might be topics or message types rather than direct physical links.

Given their complexity, each neuron-node is more comparable to a small agent in a multi-agent system or a micro-service, rather than a single math neuron in a neural net. This is a deliberate design to leverage existing AI advances (LLMs, tool use) at the lowest level of our "brain." It's as if each neuron in a biological brain had a mini-brain of its own (which is biologically unrealistic, but computationally we can grant our neurons quite some power). The challenge then is to

coordinate these powerful neurons effectively – which is where the surrounding structure (suborgans, memory, signals) comes in.

Memory Layers and Shared Memory

Memory in DBIAN exists at multiple layers:

- **Local (Neuron) Memory:** Each neuron's LLM has an internal state or context (e.g., it can hold a conversation history or have some working memory through prompts). Additionally, during a task, a neuron can store variables or results in its VM environment. However, this memory is private to the neuron unless the neuron decides to share it. A neuron might, for instance, remember the outcome of the last tool it called, or keep a cache of recent facts it used.

- **Suborgan Shared Memory:** As noted, each suborgan provides a shared memory accessible to its members. This acts as a *short-to-medium term memory* for cooperation. For example, suppose the task is to plan a route on a map: multiple planning neurons might write candidate route segments or constraints to the shared memory so that others can refine or check them. The shared memory can also serve as a *repository of best practices* within that suborgan – e.g., the evolutionary controller might log which neuron (or which genome) was most successful on recent tasks, so that others can in some way use that information (or so that parents are chosen accordingly during reproduction).

- **Global Memory:** Though not explicitly mentioned in the problem statement, one can imagine a global memory accessible via the coworking interface or main entity. This would be analogous to very widely shared knowledge (like general facts, or a world model). If implemented, global memory would allow any suborgan to deposit information that might be useful system-wide. This could be implemented as another MCP server (like a central database).

- **Episodic vs Semantic Memory:** We can classify the content of memories. Some suborgans might maintain *episodic memory* (records of specific events or tasks the system solved, akin to an experience replay), while others hold *semantic memory* (refined knowledge distilled from experiences). For instance, a memory suborgan could over time build a knowledge graph or vector store of facts that any neuron can query (similar to how the human brain consolidates episodic events into semantic memory during sleep, etc.).

Memory management within a suborgan is partly handled by the evolutionary controller and partly by the neurons themselves. We plan for suborgans to track metadata about memory usage: which items were accessed frequently, which were ignored, how memory correlates with task success. When memory capacity is limited, suborgans may implement a forgetting mechanism (e.g., least-recently-used removal or a decay of rarely used entries). Mathematically, one can imagine each memory entry having a utility value that gets updated every time the entry

is used successfully, and entries falling below a threshold get pruned. (We will present a simple model in the Mathematical Models section.)

An important aspect is that suborgans also remember **the history of configurations** and **reproduction**. For example, a suborgan might keep a log: *"Genome X + Genome Y produced Genomes Z1..Z4 at time T, of which Z3 turned out highest performing"*. This evolutionary memory can guide future breeding (perhaps avoiding repeating combinations that failed, or preferring those that led to improvements). It also allows analysis of lineage – akin to a genealogy tree of the artificial neurons.

Communication and Coworking Interface

Neurons communicate using messages that are categorized as *excitatory*, *inhibitory*, or *modulatory*. In practice, all these are messages (e.g., text or data packets) but they carry a tag indicating their type and intended effect:

- **Excitatory message:** intended to stimulate action in the recipient. For instance, a neuron might send an excitatory signal to another with some content like "Idea A seems promising, expand on it." The receiving neuron, if it respects excitatory inputs, might incorporate that suggestion into its processing (like adding a bias towards that idea).

- **Inhibitory message:** intended to suppress or caution. E.g., "Don't consider option B, it's invalid." A neuron receiving this as inhibitory input might downweight or remove option B from its considerations.

- **Modulatory message:** intended to adjust the state or parameters of the recipient, rather than direct content. For example, a modulatory signal might say "Increase your cautiousness" which a neuron might interpret as lowering its generation temperature or being more risk-averse in decisions. Another modulatory signal might carry a reward/punishment value to reinforce learning ("The last action succeeded, remember that pattern"). These are analogous to neuromodulators in the brain that influence learning rates or excitement (like dopamine signaling reward, causing neurons to adjust synaptic strengths).

Within a suborgan, neurons could potentially have direct links to each other, but a simpler implementation is that they all share the suborgan's memory and possibly subscribe to certain message topics. We can imagine that excitatory/inhibitory signals are often directed to neurons of the same suborgan (short-range connections), whereas modulatory signals might be more diffuse and possibly cross suborgan boundaries (like a general alert or context reset).

Coworking Interface: This is the central hub through which suborgans interact. It's termed "coworking" to suggest a collaborative workspace. In implementation, it could be a publish/subscribe messaging bus or a shared blackboard where suborgans post requests and results. For example, if the Planning suborgan needs a piece of information that the Memory

suborgan holds (say a fact or an image), it can send a query via the coworking interface. The Memory suborgan's neurons monitoring the interface will pick it up, respond with the info, and the interface relays it back. The coworking interface thus routes messages between suborgans as needed.

The coworking interface can also handle conflict and integration: e.g., multiple suborgans might contribute different suggestions for an output, and a mechanism at the interface (or main entity) could reconcile them (like a "voting" or selection). It can be extended to allow **multi-suborgan meetings** – for instance, the interface could initiate a synchronized session where neurons from various suborgans come together to jointly solve something (analogous to cross-functional teams). However, in most cases, the interaction is asynchronous and mediated by messages.

An important design aspect is preventing communication overload: not every neuron should broadcast everything to all others. The interface can enforce that only salient or high-level information is shared across suborgans. Lower-level chatter remains within suborgans (which is similar to how in the brain, lots of neuron-to-neuron signaling is local, and only certain summary signals travel along long-range tracts or via neuromodulators broadly).

Reproduction and Lifespan of Nodes

DBIAN introduces a **genetic reproduction cycle** for neuron-nodes:

- Each neuron-node has a **lifespan** measured in number of tasks (or cycles) it can perform. This could be a fixed number (e.g., 100 tasks) or dynamically adjusted by the evolutionary controller. When the neuron has completed that many tasks, it "ages out" and is marked for retirement. It can finish its final task and then will be removed from the active set.

- Periodically or when certain conditions are met, the evolutionary controller of a suborgan will select two parent neurons from that suborgan to undergo reproduction. Selection might be fitness-based (e.g., pick the two that solved the most tasks successfully or the ones with highest performance metrics) or sometimes include a random component to maintain diversity. We may also incorporate speciation at the suborgan level: if neurons have very different "genomes", we might ensure mating occurs between compatible or suitably diverse pairs as appropriate (mirroring NEAT's speciation preserving innovationen.wikipedia.org).

- **Mating process:** The two parent neurons will combine their "genetic" information to produce offspring. The genome of a neuron encapsulates all information needed to instantiate a similar neuron. We will detail genome structure shortly, but it includes things like model parameters or identifiers, preferred tools, connection patterns, etc. A crossover operation is applied to the parent genomes: parts of the genome from parent A and parent B are mixed. For example, if we represent the genome as a set of genes (which could be binary strings, real-valued parameters, or symbolic configurations), we might randomly choose each gene for inheritance from either parenten.wikipedia.org.

We ensure that critical components are inherited in a compatible way – using techniques akin to NEAT's innovation numbers or aligning homologous genes could be considered, though in our case genomes might be more modular (e.g., one section for LLM hyperparameters, one for tool preferences, etc., which can be recombined).

- After crossover, each child genome is subjected to **mutation**: with some probability, small changes are introduced. A mutation could be flipping a bit (if a gene is binary), randomizing a parameter slightly (for continuous values), or altering a connection (e.g., adding/removing a link to a suborgan). Mutation rate and distribution are parameters managed by the evolutionary controller, possibly adapting over time.

- The result is a set of offspring genomes. According to the problem statement, we specifically create **4 new neurons from 2 parents** (so each mating yields 4 children). This is a design choice to amplify successful genetic combinations and also rapidly explore variations. It's akin to giving the offspring a "litter." Those 4 new neurons are then instantiated as active neurons in the suborgan. The parents might remain as well (especially if they still have lifespan left and continue to be useful), or we might retire the parents at this point to keep population size in check. There are multiple strategies here: one could maintain a steady population by retiring parents when offspring come, or allow some growth and then cull least fit later.

- Each new neuron is assigned a fresh lifespan counter and becomes part of the working population, ready to take on tasks.

Genome Structure: We now illustrate a simplified view of what a neuron's genome might look like, using a YAML-like format for clarity:

yaml
CopyEdit

```yaml
# Example of a neuron-node genome in YAML format
neuron_id: 47              # unique identifier
suborgan: "Memory"         # which suborgan it belongs to
model:
  base: "GPT-4-mini"       # identifier of base LLM model
  fine_tune: "memory-specialist-v2"  # optional fine-tuning ID
  temperature: 0.7         # decoding hyperparameter
modalities: ["text", "vision"]  # modalities the neuron can handle
tool_preferences:          # MCP tools this neuron is skilled at
  - name: "WebSearch"
    usage_bias: 0.9        # prefers to use this frequently
  - name: "Calculator"
    usage_bias: 0.5
```

```
connections:
  excitatory_targets: [51, 52]    # IDs of neurons it excites
  inhibitory_targets: [60]        # IDs of neurons it inhibits
  modulatory_targets: []          # no modulatory targets in this
example
lifespan_limit: 100         # tasks it can do before retirement
fitness_score: 0.85         # performance metric (computed during life)
parent_ids: [12, 15]        # IDs of parent neurons (for lineage
tracking)
mutation_rate: 0.01         # (could be uniform or gene-specific)
```

Listing 1: A hypothetical genome for a neuron node. In this example, the genome encodes the suborgan assignment, the model and hyperparameters (it's using a hypothetical "GPT-4-mini" model fine-tuned for memory tasks, with a certain creativity setting), the modalities (it can process text and vision), preferences for using two tools (with biases indicating how strongly it tends to use them), connections to other neurons by ID (perhaps within its suborgan or even cross-suborgan if allowed), its lifespan setting, a placeholder for fitness score (which is not part of the genome to inherit per se, but kept for selection purposes), and lineage info (who its parents were). Not all of these would necessarily be directly encoded for crossover (for instance, `fitness_score` might not carry over; it's more for selection). But parameters like model choice, fine-tune, and connections definitely would be subject to crossover/mutation.

During crossover, we might split the genome by sections. For example, one child might get the model parameters from parent A and tool preferences from parent B, while another child gets the opposite. Connections could be merged by taking the union or intersection of parent links plus some random additions/removals. Parent IDs of course would be set to the actual parents.

When a neuron is created from a genome, it loads the specified model (or a fresh copy of it) and configures itself accordingly. Note that large components like the base model weights are not mutated directly (we don't expect to evolve raw weights at the bit level – that would be infeasible given modern model sizes). Instead, mutation might toggle which base model is used from a small set, or adjust numeric hyperparameters. It's also possible to encode some **memory** or knowledge into the genome (for instance, the fine-tune name could imply inherited knowledge). In future implementations, we might even allow neurons to pass on some learned memory to offspring, a kind of Lamarckian inheritance (which is an interesting deviation from pure Darwinian evolution). For now, we assume inheritance is mostly of design/architecture/hyperparameters, not the exact learned weights (though offspring starting from same base model can learn similar knowledge during their life).

Lifespan and Retirement: The lifespan mechanism ensures continuous turnover in the population. A neuron's lifespan decrement could happen per task or per time unit. Retirement involves deregistering the neuron from the suborgan (so it no longer receives tasks or

messages) and freeing its resources. We may allow a retiring neuron to dump some of its knowledge into the shared memory before exiting (like “uploading” what it learned or summarizing its experiences), so that its contributions aren’t lost. This is analogous to how old members of a society might mentor or leave records for the next generation.

Putting It Together

When DBIAN is running, each suborgan will be concurrently solving sub-tasks with its neuron-nodes. They communicate locally (excit/inhib signals influencing which neurons take lead, modulatory signals adjusting parameters) and share memory. When needed, suborgans exchange information globally via the coworking interface. Meanwhile, in the background (or during idle times or after task completion), the evolutionary controller will spawn new neurons and retire old ones, based on performance logs and lifespans. Over time, we expect each suborgan’s population to become more efficient at its specialized task – akin to a micro-evolution optimizing memory-handling neurons in one suborgan, planners in another, etc. The global behavior of DBIAN emerges from the interplay of these evolving, specialized parts.

In the next section, we outline the methodology of how DBIAN operates step-by-step in a typical scenario, and then formalize aspects of the communication and evolution with mathematical models.

Methodology

We now describe how the DBIAN system functions dynamically. This includes how tasks are processed, how suborgans coordinate via the coworking interface, and how the evolutionary cycle is integrated into the agent’s operation.

Task Processing Workflow

1. **Task Ingestion:** The main DBIAN agent (main entity) receives a task or input. This could be a user query (in an interactive QA system), a sensory input frame (in a robotics context), or an internally generated goal. The main entity analyzes the task at a high level to determine which suborgans need to be involved. For example, if the task is “Describe how to tie a shoelace,” the main entity might engage the Memory suborgan (for any known instructions), the Planning/Reasoning suborgan (to formulate a clear step-by-step explanation), and a Language suborgan (if one exists specifically to refine output phrasing). It would not, say, engage a Vision suborgan since this task is purely textual.

2. **Suborgan Activation:** The main entity triggers those suborgans by sending an initial message or placing an entry in the coworking interface indicating the task context. Each relevant suborgan then activates its internal process. Activation might mean the suborgan’s neurons are given a goal or query related to the task. For instance, the Planning suborgan might get the prompt “Plan steps for tying a shoelace,” the Memory

suborgan might get “Recall any known procedures for shoelace tying,” etc.

3. **Intra-Suborgan Processing:** Within each active suborgan, neurons work (mostly in parallel) on their piece of the problem. How they divide work can vary:

    - In some cases, they may compete or cooperate. A possible approach is a **broadcast-and-competition**: the suborgan’s shared memory holds the current best solution attempt, and neurons attempt improvements. Excitatory signals can encourage certain neurons to contribute, while inhibitory signals can suppress redundant or unhelpful ideas.

    - For instance, one neuron in Planning might propose a set of steps and write it to shared memory. Another neuron reads it, sees a flaw, and via an inhibitory message suggests removing or altering a step. Yet another neuron might receive an excitatory cue to expand a particular step with more detail.

    - The suborgan’s evolutionary controller (or any supervisory logic) might ensure not all neurons activate at once to avoid chaos – possibly a scheduling mechanism or simply by the nature of signals some neurons wait until they are excited. This is analogous to how in a brainstorming session not everyone talks at once, but individuals contribute in turn.

    - Eventually, the suborgan arrives at an *output* for the current cycle: e.g., the Planning suborgan finalizes a sequence of steps.

4. **Inter-Suborgan Coordination:** Throughout the process, suborgans exchange information via the coworking interface:

    - The Memory suborgan might return a factual instruction sequence it remembered (“Loop the lace, pull through…”). This gets posted to the interface.

    - The Planning suborgan picks it up and integrates it into a coherent plan.

    - Perhaps a Language suborgan (if present) later takes the raw plan and rephrases it nicely.

    - If a conflict arises (say two suborgans propose different answers), the main entity or a designated arbitration suborgan reconciles it. This could be done by a simple rule (prefer plan with higher confidence) or by a joint discussion (one suborgan might ask another for justification via the interface).

    - The coworking interface may also handle timing – e.g., it can signal all suborgans when a final answer is needed, prompting them to wrap up.

5. **Task Completion:** The main entity collects the outputs from suborgans and composes the final result to output externally. In our example, it might take the refined step-by-step instructions from the interface (with language polishing done) and present that as the answer to the user. In a robotic action scenario, it might take a decision from a Decision suborgan and execute it on the robot.

During this task-focused processing, **neurons use MCP tools as needed**. For instance, if a neuron in the Memory suborgan didn't have the needed info on shoelace tying, it might call a WebSearch tool via MCP to find a wikiHow article, then parse it and share key points in memory. The use of tools is guided by the neuron's own policy (which can be evolved). One neuron might prefer using external knowledge, another might rely on internal knowledge. Over time, presumably the evolutionary process favors whichever approach yields better results for that suborgan's tasks (and likely it will be a combination of both).

Evolutionary Cycle Integration

The evolutionary mechanism operates in parallel with task processing. We design it to not disrupt tasks; ideally, evolution happens in the "background" or in between tasks:

- Each neuron has a lifespan counter. Each time it completes a task (or a unit of work contributing to a task), the counter decreases. The evolutionary controller monitors these.

- When a neuron's counter reaches zero, the controller flags it for retirement. It may allow it to finish any current contribution, then gracefully shut it down. Before removal, the neuron might dump any useful info to shared memory (e.g., "I found method X effective in solving Y"). That info can later be picked up by offspring or others.

- The controller decides when to initiate reproduction. This could be:

    - **Periodic:** e.g., after every N tasks handled by the suborgan, spawn some new neurons.

    - **On Retirement:** whenever a neuron retires, immediately breed new ones to replace it (this could even be two retirements triggering one mating to yield four children, thus expanding population by net +2).

    - **On Opportunity:** if two neurons have shown exceptional synergy or performance, the controller might proactively mate them even if they're not expired – to quickly propagate their traits. This is akin to seizing a good combination early.

- Selection of parents takes into account each neuron's *fitness*. Fitness can be a composite measure: number of tasks solved successfully, quality of contributions

(perhaps rated by the main entity or by outcome metrics), and efficiency (speed, resource usage). The highest fitness nodes are prime candidates. However, diversity is also important to avoid premature convergence. We might use a roulette wheel selection or tournament selection that sometimes picks a less fit but genetically distinct neuron. Also, to simulate speciation, the controller could cluster genomes by similarity and ensure mating happens within clusters or picks parents that are not too genetically distant (to avoid many broken offspring).

- Once parents are selected, the reproduction yields offspring as described. The offspring neurons are initialized and integrated into the suborgan:
    - They register with the suborgan's communication and memory systems.
    - They may start with an "orientation" period: e.g., read the shared memory to get up to speed on current context, or shadow a task to calibrate. Or they might be immediately thrown into the mix for a fresh task to see how they do.
    - Offspring could also potentially inherit some memory from parents – for example, the shared memory log could have entries like "Parent 12 learned that tool A is useful for Q". If the genome encodes parent IDs, the offspring might parse the memory for any notes from those parents to bootstrap themselves.
- The controller may impose a limit on total neurons. If population would grow too large, it might either reduce reproduction frequency or intentionally retire some neurons early (perhaps the poorest performers) to free slots. This keeps computational load manageable. It's similar to fixed population size in GA – some selection method to drop the weakest when new ones come in.

This evolutionary process is essentially a continuous online genetic algorithm operating within each suborgan. It bears similarity to rtNEAT's method where networks are continuously evaluated and replaced without halting the systemen.wikipedia.org. DBIAN thus maintains a sort of **meta-learning loop**: at one level, neurons (with their LLMs) learn and adapt within tasks (e.g., an LLM fine-tuning itself or updating its prompt strategies during interactions, analogous to synaptic plasticity), and at another level, less frequently, neurons themselves are replaced by new ones (analogous to a population evolving).

Example Scenario

To make this concrete, consider a scenario where DBIAN is an AI personal assistant that over time learns to better serve its user:

- Initially, suborgan A (Memory) has neurons that search the web for answers, and suborgan B (Reasoning) has neurons that try to reason through problems. Over many Q&A sessions, the controller notices two Reasoning neurons consistently provide

accurate answers and decides to breed them. The offspring get a mix of their traits: one offspring might inherit the tool-using propensity of parent1 and the logical style of parent2. These new neurons perhaps are even better, and the suborgan's performance on difficult questions improves.

- Another suborgan C (perhaps "Motivation/Prioritization") evolves neurons that manage what the assistant should do when multiple user requests come in. If a neuron makes poor choices (user is dissatisfied with what it prioritized), its fitness is low and eventually it's replaced by offspring of better neurons that handled priorities well. Over time, suborgan C's population converges to neurons that handle prioritization in a way tuned to the user's preferences (e.g., always address urgent queries first, etc.).

- Meanwhile, the Memory suborgan's shared memory accumulates a knowledge base of the user's favorite things. Neurons in Memory that effectively store and retrieve these preferences become highly valued. The evolutionary controller might keep those around longer or give them more offspring.

- If the user's needs change (say now the user asks more coding questions than before), the system can adapt: perhaps a new "Coding" suborgan could be spawned (this would be a larger structural evolution, which is an extension we could consider), or more likely the Reasoning suborgan evolves some neurons that are good at coding (maybe by spawning ones that use a Python tool extensively). There might be a mutation that adds a connection from a neuron to a new tool (e.g., a code interpreter), and if that helps answer coding questions, that neuron's fitness jumps and it propagates this trait.

Throughout this, the user hopefully experiences an assistant that is getting smarter and more attuned, without explicitly knowing that under the hood a population of "digital neurons" are competing and reproducing.

Modular Component Interactions

We emphasize how the components described in the Architecture section come together:

- The **Main Entity** orchestrates at high level but doesn't micromanage. It invokes suborgans and trusts them to do their job, and it handles final output composition.

- **Suborgans** manage internal complexity (like mini-agents within the agent). They expose an interface (via coworking) to request or provide info.

- **Neurons** carry out actual computations. They are largely autonomous – each decides when to send signals or write to memory based on its programming (which is partly from its LLM, partly from any coded policy).

- **Shared Memory** in each suborgan is accessed by neurons through reads/writes – perhaps via an API call in their VM (which could ensure concurrency control).

- **Coworking Interface** might be implemented as a message queue where suborgans subscribe to certain topics. When suborgan A needs something from B, it publishes a message “Request: X” tagged for B, and B’s listener picks it up. The interface also could log all inter-suborgan communication for analysis.

- **Evolutionary Controller** can be thought of as a background thread in each suborgan that wakes up at certain intervals, checks the state (fitness of neurons, any that died or retired), and triggers reproduction as needed. It updates a *Suborgan Memory of Evolution* (noted in architecture as tracking configurations and reproduction history). This could be stored in the shared memory or separately. For instance, after each breeding, it might store a record: (parent1_id, parent2_id, children_ids, context_of_breeding, initial performance metrics of children). Over time, it can analyze which parent pairs were most fruitful – akin to learning a good breeding strategy (perhaps certain combinations yield consistently good offspring, etc.).

The methodology ensures that **evolution and task-solving inform each other**: good performance on tasks increases reproductive success, and conversely the evolutionary changes produce (hopefully) even better task performance in the future. This feedback loop embodies an *optimization process* with respect to the goals the system is tasked with, albeit a very complex one. We rely on emergent evolution to handle aspects that are hard to explicitly program (like finding the optimal prompt or deciding which combination of tools yields the best result), by allowing variation and selection to gradually discover high-performing configurations.

In the following section on mathematical models, we formalize some parts of this process to give more precision to these descriptions.

Mathematical Models

In this section, we introduce mathematical formalisms for three core aspects of the DBIAN framework: **neural communication dynamics** (excitatory/inhibitory/modulatory signal integration), **genetic reproduction** (crossover and mutation of neuron genomes), and **suborgan memory management** (tracking and updating knowledge of efficient configurations). These models are abstractions intended to capture the essence of how the system operates, providing a theoretical foundation.

Communication Model

Each neuron-node can be seen as a processing unit that takes inputs (from other neurons or environment) and produces output signals. We denote by $O_i(t)$ the **output** of neuron i at time (or cycle) t. In an LLM-based neuron, $O_i(t)$ might represent a particular message or

result the neuron produces (e.g., the content it writes to shared memory or sends via coworking interface) during cycle t. For modeling purposes, we can consider $O_i(t)$ as a numeric activation or utility value representing the significance of neuron i's output at that time.

Neuron i receives inputs from potentially many other neurons. Let E_i be the set of neurons with excitatory connections to i, I_i the set with inhibitory connections to i, and M_i the set with modulatory connections to i. These sets are defined by the network's connectivity (which evolves over time). Correspondingly, each connection has a weight (or efficacy) that determines how strongly it influences i. Let $w_{j \to i}^+$ denote the weight of an excitatory connection from neuron j to i, $w_{k \to i}^-$ the weight of an inhibitory connection from k to i, and $w_{\ell \to i}^m$ the weight of a modulatory connection from ℓ to i.

We can then write the **net input** to neuron i at time t as:

Iinet(t)=∑j∈Eiwj→i+ Oj(t) − ∑k∈Iiwk→i− Ok(t) .I_i^{net}(t) = \sum_{j \in E_i} w_{j \to i}^+ \, O_j(t) \;-\; \sum_{k \in I_i} w_{k \to i}^- \, O_k(t) \,. Iinet(t)=j∈Ei∑wj→i+Oj(t)−k∈Ii∑wk→i−Ok(t).

This formula states that i sums up all excitatory inputs (each contributing their output times the excitatory weight) and subtracts all inhibitory inputs (each contributing their output times an inhibitory weight). Modulatory inputs are not summed in the same way; instead, they influence parameters of neuron i's response. We introduce a modulatory factor $M_i(t)$ that affects neuron i at time t:

Mi(t)=∑ℓ∈Miwℓ→im Oℓ(t) .M_i(t) = \sum_{\ell \in M_i} w_{\ell \to i}^m \, O_\ell(t) \,.Mi(t)=ℓ∈Mi∑wℓ→imOℓ(t).

Think of $M_i(t)$ as a context or gain factor. For instance, a positive $M_i(t)$ might lower i's firing threshold or increase its responsiveness, whereas a negative $M_i(t)$ could raise the threshold (making i less likely to act).

We then define neuron i's output based on these:

Oi(t)=F (Iinet(t); θi+Mi(t)) ,O_i(t) = F\!\Big(I_i^{net}(t); \; \theta_i + M_i(t) \Big) \,,Oi(t)=F(Iinet(t);θi+Mi(t)),

where $F(x; \theta)$ is a neuron activation function with parameter θ (which could be threshold or other bias). In a simple case, if we treat neuron i as a binary activator: $O_i(t) = 1$ if $I_i^{net}(t)$ exceeds $\theta_i + M_i(t)$, else 0. But in our LLM context, F could be more abstract – e.g., F could represent whether neuron i decides to contribute to the shared memory or remain silent. A high excitatory input might prompt it to speak up, whereas inhibitory input might silence it. The modulatory input $M_i(t)$ could dynamically adjust θ_i (the threshold) making it easier or harder to activate.

Alternatively, if we interpret $O_i(t)$ as not just binary but a continuous measure of contribution strength, F could be an identity (linear) or a sigmoid: $O_i(t) = \sigma(I_i^{net}(t) - (\theta_i + M_i(t)))$ for some sigmoid σ. For simplicity, one can assume each neuron has a

baseline threshold θ_i for activation and modulatory input effectively adds or subtracts from that threshold.

Interpretation:

- If neuron j sends an excitatory message with magnitude $O_j(t)$ and $w_{j\to i}^+$ is large, neuron i receives a strong push to activate (e.g., a planning neuron strongly suggesting another planning neuron to consider a path).

- If neuron k sends an inhibitory message, it provides a subtractive influence. For example, if $I_i^{net}(t)$ becomes negative or falls below threshold due to inhibitory inputs, neuron i might not produce output (like a memory neuron suppressing another's recall to avoid redundancy).

- A modulatory message (say from neuron ℓ) might not carry direct content but can cause $M_i(t)$ to be positive, effectively lowering θ_i. This could represent a signal like "be more active," resulting in neuron i being more likely to output even with smaller I_i^{net}. Conversely, a negative modulatory $M_i(t)$ raises the bar, telling i to be cautious or silent unless it has a very strong net excitatory input.

This model captures the essence of how multiple signals integrate. In a running DBIAN, these computations might not be explicit numeric calculations (continued)

This model captures the essence of how multiple signals integrate. In a running DBIAN system, these calculations might not literally occur as equations, but the behavior of the messaging system can be **interpreted** in these terms. For instance, a neuron's decision to output or stay silent can be seen as comparing an internal utility (based on received excitations/inhibitions) against a threshold modulated by context. The modulatory signals correspond to dynamic adjustments – e.g., a "focus" signal could globally lower thresholds in a suborgan to make neurons more eager to contribute at a critical moment, much like a surge of dopamine increasing neural activity in certain brain circuits.

This formalism also helps when designing or tuning the system. One could assign numerical weights to connections and implement a simplified **activation update rule** as above to decide which neurons get to write to shared memory next, etc. It provides a way to reason about stability (too many excitations could lead to everyone talking at once – akin to an epileptic seizure in a network sense – whereas too many inhibitions could silence the system). Tuning the balance of excitatory and inhibitory influences is crucial for coherent behavior, just as in biological neural networks a balance is needed for stable dynamicazoai.com】.

Reproduction (Genetic Algorithm) Model

We model each neuron-node's *genome* as a vector of genes or parameters: $G = (g_1, g_2, \ldots, g_L)$. The length L and contents of this vector depend on our genome design (as

illustrated in the YAML example earlier). For simplicity, assume we have a fixed-length encoding for key traits (some genes might be binary flags for tool preferences, some might be numeric hyperparameters, others might be identifiers for model type, etc.). The **fitness** of a neuron i, denoted F_i, is a measure of its performance over its lifetime (higher is better). This could be a weighted sum of task success rates, contribution quality, etc., accumulated until it reproduces or dies.

When the evolutionary controller selects two parent neurons A and B for mating, they have genomes G^A and G^B. We perform **crossover** to produce an intermediate offspring genome G^{AB}. A simple crossover strategy is a gene-wise random mix (sometimes called uniform crossover):

gkAB={gkA,with probability 0.5,gkB,with probability 0.5,for each gene k=1…L.g^{AB}_k = \begin{cases} g^A_k, & \text{with probability } 0.5, \\ g^B_k, & \text{with probability } 0.5, \end{cases} \qquad \text{for each gene } k = 1 \ldots L.gkAB={gkA,gkB,with probability 0.5,with probability 0.5,for each gene k=1…L.

In other words, the child randomly inherits each gene from one of the two parents with equal chance. (More sophisticated crossovers could be used, such as 1-point or 2-point crossover if genes have an order, or blending for numeric values, but uniform gives a baseline.) This yields a combined genome G^{AB} that is a mix of A and B.

Next, we apply **mutation** to G^{AB}. Define a mutation operator $\mu(\cdot)$ that perturbs a gene. For each gene k:

- With probability p_{mut} (the mutation rate), we set $g^{child}_k = \mu(g^{AB}_k)$.

- With probability $1 - p_{\text{mut}}$, we leave it $g^{child}_k = g^{AB}_k$.

The mutation $\mu(g)$ could be defined in various ways depending on gene type:

- If g is binary or a categorical choice, $\mu(g)$ might flip the bit or switch to a different category (possibly with some bias).

- If g is numeric (like 0.7 for a temperature setting), $\mu(g)$ might add a small random delta drawn from a distribution (e.g., Gaussian with mean 0 and small standard deviation) and clamp the result within valid bounds.

- If g is an identifier (like which base model to use), $\mu(g)$ might randomly choose a different valid identifier from the set (simulating a random mutation that swaps the model).

We can incorporate *gene-specific mutation rates* if needed; for now assume a uniform p_{mut} for all genes for simplicity.

The outcome of crossover+mutation is one **offspring genome** G^{child}. To produce four offspring, we can repeat this process four times (each time may result in slightly different mixes and mutations due to randomness). Denote the four children's genomes as G^1, G^2, G^3, G^4. These are then instantiated as new neuron-nodes in the suborgan.

If parents A and B have high fitness, typically their children will start with a good combination of traits, but mutation ensures new variations also appear. Some offspring might not perform well (that's expected; not all children of great parents are great). The idea is that by generating multiple offspring, we increase chances that at least one or two will exceed the parents by getting a lucky combination of their complementary strengths. The **selection pressure** in the system comes from the fact that only neurons which survive and perform (i.e., avoid retirement and perhaps are chosen again as parents) will propagate their genes further. Over many generations, one would expect convergence to sets of genes that are well-suited to the tasks the suborgan handles.

To formalize selection a bit: when picking parents, one could use a probability proportional to fitness. For example, if suborgan S has population ${1,2,\ldots,N}$ with fitness $F_1, \ldots, F_N$, we might pick parent A with probability $F_A/\sum_{i\in S} F_i$ (roulette selection), and similarly for B (ensuring $B \neq A$). Alternatively, a **tournament selection** of size k could be used: pick k random neurons and choose the best among them as A; repeat for B. These methods are standard in genetic algorithms to probabilistically favor the fitter individuals while still giving others a chanc[en.wikipedia.org]. DBIAN's evolutionary controller can adopt any such method. Speciation, if implemented, means we would restrict selection to within a species (e.g., within clusters of similar genomes) to maintain diversit[en.wikipedia.org]. In practice, suborgans themselves could be seen as species, or if a suborgan's internal population diverges (some neurons with very different strategies), the controller might avoid mating those too-different ones and rather mate like with like or ensure innovation survives by not always mating only the top two (which might be very similar).

Continuous Evolution: We do not have discrete generations in DBIAN. The above process happens whenever triggered by the controller. We can model it in discrete steps for analysis: say every T tasks or time units, a mating event happens. Over time, let's denote by $P(t)$ the set of active genomes at "epoch" t. The evolutionary process is:

$$P(t+1)=(P(t)\setminus D(t)) \cup B(t), P(t+1) = \big(P(t) \setminus D(t) \big) \;\cup\; B(t), P(t+1)=(P(t)\setminus D(t)) \cup B(t),$$

where $D(t)$ are the genomes of neurons that died/retired in the interval (removed from population) and $B(t)$ are the genomes of newly born neurons in that interval. The size of $B(t)$ depends on how many reproduction events happened. In steady-state, one might have the number of births equal the number of deaths to keep $|P|$ roughly constant. If each reproduction yields 4 and we retire 2 (the parents or others) at that time, net +2, we might later

retire more to compensate. Over a long run, assume population size N is regulated to stay around some target.

We can also consider an evolving **gene pool** in the suborgan memory. Let $\mathcal{G}(t)$ represent the multiset of all gene values present in the population at time t. We could track frequencies of gene variants (alleles) and examine how they change – akin to population genetics. For example, if gene k is "uses tool X or not", we might see the allele "uses tool X" go from 30% to 80% frequency over time if tool X turns out very useful. This is another way to quantify the system's adaptation.

Suborgan Memory and Knowledge Management Model

Each suborgan's memory and logs help guide its internal processes and future evolution. We outline a few elements that suborgans track and how they can be modeled:

- **Efficient Configuration Records:** A configuration could refer to a particular arrangement or state of the suborgan's network that proved effective. For instance, "Neuron 5 strongly active, Neuron 7 suppressed, using Tool A frequently" might be a configuration pattern that yielded high reward. The suborgan can store a set of top configurations $C_{\text{best}} = {c_1, c_2, \ldots, c_m}$ with associated performance metrics $U(c_i)$ (utility scores). When new outcomes occur, the suborgan updates this:
    - If a new configuration c_{new} achieved utility $U(c_{\text{new}})$, and if $U(c_{\text{new}})$ is higher than the lowest in C_{best}, it can be added (possibly evicting the worst). This is like maintaining a **priority queue** of best scenarios.
    - The utility $U(c)$ might be defined as a weighted sum of accuracy, speed, resource cost, etc., depending on what "efficient" means (likely task reward minus some penalty for resources).
    - This record can inform decisions: e.g., the evolutionary controller might attempt to recreate or sustain a known good configuration by favoring certain genes or by sending modulatory signals to push the current state towards a recorded good state.
- **Communication Flow Patterns:** The suborgan can log sequences or graphs of message exchanges that led to successful task completion. For example, it might note that when neuron X excited Y and then Y inhibited Z, the outcome was good. Over time, it can derive a pattern like "$X \rightarrow Y$ (excitatory) followed by $Y \Leftarrow Z$ (inhibitory) is a useful motif." We could formalize this as a frequency count of n-gram patterns in communication:
    - Let's define a "communication event" as $(i \xrightarrow{\text{sig}} j)$ meaning i sent a signal of type sig to j. We can count occurrences of sequences like

$(i \to j), (j \to k)$ and measure correlation with success.

    - The memory might store a weighted directed graph G_{comm} where nodes are neuron IDs (or roles) and edge $(i \to j)$ has a weight representing how often that communication (with whatever signal type) appears in successful episodes minus unsuccessful ones. This effectively learns a subnetwork of *useful communication links*. If G_{comm} shows a particularly strong link, the controller might ensure future neurons preserve that link (when breeding, favor keeping that connection).

    - Conversely, if some communication flow is frequently associated with failures (e.g., a certain neuron always sends a distracting excitatory signal that leads others astray), that might appear as a negative weight, and the controller could decide to remove or weaken that link (perhaps by mutating the offending gene that causes it).

- **Tool Use Statistics:** For each external tool (or MCP server function) T, suborgan memory can maintain stats like:

    - n_T = number of times tool T was invoked by any neuron.

    - s_T = number of successful outcomes from using T (success could be defined as tool returned useful data that contributed to solving task).

    - We can define an empirical success probability $\hat{p}_T = \frac{s_T + \alpha}{n_T + \alpha + \beta}$, using a Beta prior (α,β) to avoid zero-division (this is a Bayesian estimate). For instance, with $\alpha=\beta=1$ (uniform prior), $\hat{p}_T = \frac{s_T + 1}{n_T + 2}$.

    - The suborgan can prefer tools with higher $\hat{p}_T$. If a neuron has a choice of calling either a web search or a local database, and the memory knows web search success rate is 90% vs database 50%, it might excite neurons that tend to use web search.

    - Tools also have a usage cost; memory might store average time or tokens used per tool call, enabling a trade-off analysis (a fast but slightly less accurate tool might be better under time constraints).

    - When mutation introduces a new tool usage (a gene flips from “not using T” to “using T”), the initial success might be low, but if it occasionally yields a big improvement, the stats will update and could prove its worth.

- **Reproductive History:** The suborgan logs each reproduction event: $(id_A, id_B) \to (id_1, id_2, id_3, id_4)$ at time t, along with any immediate observations like parents’

fitness and initial children performance. Over time, genealogical trees or lineages can be constructed. This can be analyzed to detect phenomena like:

- *Lineage success:* e.g., “most of the current high-performers descend from ancestor node 7 about 3 generations back”. The controller could then examine what was special about that ancestor (maybe a particular gene) and ensure it’s preserved.

- *Inbreeding or loss of diversity:* if the log shows many events are between closely related individuals, genetic diversity might be shrinking. The controller might then decide to introduce a random new genome or force mating between dissimilar ones (somewhat akin to injecting novel DNA or encouraging exploration).

- *Mutation impact:* by comparing child performance to parents, the log can estimate which mutation types tended to be beneficial. For example, it might find that whenever the “model type” gene mutated to a larger model, fitness improved, suggesting that gene has not converged and maybe pushing further could help (at cost of resources). Or it might find too high a mutation rate causing many failures, and thus dynamically adjust p_{mut} downward for stability.

From a modeling perspective, we can represent the *knowledge in memory* as a set of variables updated over time. For instance:

- Let $Q(c)$ be the recorded utility of configuration c, updated as a running average when c (or something similar to c) occurs.

- Let W_{ij} be the weight for communication $i \to j$, updated as $W_{ij} \leftarrow W_{ij} + \eta ,(r - \bar{r})$ whenever $i \to j$ happened in a task with outcome r (reward) relative to average $\bar{r}$ – a kind of Hebbian credit assignment for communications.

- Let $\hat{p}_T$ update via Bayesian posterior after each use of tool T as mentioned.

- Let ΔF records for reproduction: e.g. for each event, store $F_{\text{child avg}} - \max(F_A, F_B)$, which is the difference between average child fitness and parent fitness. A positive average indicates sexual reproduction often yields better-than-parents (on average), which is good; a negative would indicate maybe something’s wrong (like epistasis issues or disruptive crossover). We could aggregate these stats.

These mathematical notions inform how one might implement learning at the suborgan level on top of evolution. The **memory acts as a critic**, guiding the evolution with additional feedback beyond raw fitness. In future iterations, this could lead to meta-learning: the suborgan might

adjust how it selects parents or what mutations to favor, essentially evolving the evolutionary strategy itself (though that is beyond our current scope).

Discussion

The DBIAN framework described here is a bold synthesis of biological inspiration and computational pragmatism. In this section, we discuss how **biological principles** influenced the design, where we **deviated or abstracted away from biology** and why, and what the implications are for achieving the goals of emergent intelligence and internal innovation. We also compare the updated DBIAN to its previous incarnation and to related systems to highlight improvements.

Biological Inspiration vs. Abstraction

We took strong inspiration from the *architecture of the brain*: modular organization, specialized regions, and multi-channel communication. Suborgans in DBIAN explicitly mirror brain regions with distinct functionxmpro.com】, which should facilitate interpretability (each cluster of neurons has a known role) and efficiency (task-specific optimization). We also mirrored the idea of **excitatory and inhibitory signaling** – crucial for brains to regulate activity – by allowing neurons to send activating or suppressing messages. The addition of modulatory signals is analogous to neuromodulators that affect learning and context (like dopamine's role in reward-based learning). By including these three types of interactions, DBIAN can, for instance, implement a reinforcement learning-like mechanism internally (via modulatory reward signals) on top of direct reasoning (excitatory flows) and pruning of bad ideas (inhibitory vetoes).

However, we made **significant abstractions**. In a real brain, neurons are simple electrochemical units; intelligence emerges from millions of them in complex networks. In DBIAN, each "neuron" is actually a high-level AI model with considerable built-in intelligence (an LLM with vast knowledge). This is a necessary departure given current technology – we cannot simulate billions of spiking neurons efficiently to get human-level cognition, but we can leverage pre-trained LLMs as a shortcut to complex cognitive functions. This could be seen as implementing a *brain at a higher level of granularity*: instead of millions of tiny neurons, we have dozens of "macro-neurons," each like a small expert agent. This choice trades biological realism for practical power; it assumes that cooperation among these macro-neurons can lead to emergent behavior analogous to cooperation among actual neurons, even if the scale and nature of components differ.

Another major deviation is the introduction of **genetic evolution within one agent's runtime**. In nature, evolution happens across generations of organisms, not within a single brain. We justify this abstraction because it provides a mechanism for **open-ended improvement**. Real brains adapt by rewiring synapses (learning); our system does allow learning (the LLMs can fine-tune or update in minor ways), but we wanted a more radical adaptability – the ability to create entirely new components (new neurons) and new structures (new connections) on the fly. Biological evolution is one of the few processes known to produce truly novel structures and

behaviors. By transposing it to run inside the agent, we hope to capture some of that innovative power. One can think of this as *simulated evolution* or *continual structural learning*. It is admittedly a strong abstraction – it's as if each person's brain could spawn new neurons with new random DNA throughout life, which real brains largely avoid (aside from limited neurogenesis). But some analogies exist: the brain's immune cells (microglia) might mutate to fight infections, or one might consider ideas evolving in a "marketplace" of mind. We use the evolutionary algorithm because it is a well-established method for searching complex spaces without gradient information, and our problem – finding better multi-agent configurations – is exactly such a space.

We also abstracted the *timescales*: in DBIAN, one "lifetime" of a neuron might correspond to minutes or hours of work, after which it reproduces or dies. This is obviously not biological (neurons live for decades). But it is computationally necessary to cycle through many "generations" to evolve effectively. By shortening the lifespan, we accelerate evolution. This is similar to how in evolutionary algorithms we compress what might be eons of natural evolution into thousands of iterations in silico. The risk is that other components (like the LLM's learned knowledge) may not fully adjust in such short spans. We mitigate that by allowing the shared memory to carry over some knowledge and by the fact that new neurons start with pre-trained models, so they're not learning from scratch.

Emergent Intelligence and Internal Innovation

One of the ultimate goals of DBIAN is to foster **emergent intelligence** – behaviors or capabilities that arise from the interactions of components, not explicitly programmed in any single component. By having multiple specialized suborgans, we encourage **diversity of thought** within the system. Each suborgan can develop its own "culture" or approach (via evolution) to problems. When they collaborate through the coworking interface, the combination can yield solutions that neither alone could achieve. For example, imagine a scenario where solving a task needs both creative generation and strict logical verification. We could have a suborgan that evolves very creative (even somewhat random) generation strategies, and another that evolves rigorous checking strategies; together, using excitatory/inhibitory exchanges, they might produce creative ideas and filter them to one that is both novel and correct. This kind of *dialectic process* could lead to emergent problem-solving ability that looks quite intelligent and was not explicitly coded, but rather emerged from the interplay and reciprocal feedback of sub-systems.

The **internal innovation** aspect refers to the system's ability to improve itself in ways not anticipated by its initial configuration. Through evolution, DBIAN can spawn entirely new "ideas" in the form of new neuron agents with novel combinations of traits. For instance, it might discover an unusual use of a tool (maybe using a drawing tool to do computation by encoding numbers as images – a contrived example, but illustrating creativity) that a human designer didn't foresee. If that proves useful, those traits propagate. In a sense, the system is conducting R&D internally: trying variants and keeping the successful ones. This is analogous to how human culture evolves techniques through individuals trying things – here it's neuron-nodes trying strategies. The expectation is that over time, DBIAN becomes not just *better at tasks it*

was given, but possibly finds *new ways to approach tasks*, making it more robust to changes and potentially capable of tackling problems beyond its initial scope.

However, achieving truly emergent intelligence is hard. Many multi-agent systems end up with agents doing fairly understandable things that we programmed them to do, just in parallel. The evolutionary component is key to go beyond that, injecting randomness and selection which can yield surprising behaviors. A known outcome in some evolutionary simulations is that agents find shortcuts or even hacks to maximize reward (sometimes against the designers' intent). We have to be mindful: internal innovation should be directed towards intended goals (hence we define fitness aligned with what we want, and modulatory signals can be used to give feedback on goal achievement). We likely will observe some **emergent specialization** – e.g., within a suborgan, different neurons might evolve to handle different sub-tasks (one memory neuron might specialize in people's names, another in dates, etc., if that improves overall performance). We might also see **emergent communication protocols** – the signals might take on meanings (like certain patterns of messages become a code for something) without us explicitly designing them. That would parallel how in the brain, groups of neurons develop representations (like a concept being represented by a firing pattern across many neurons) that are not directly engineered but arise from learning. We are essentially hoping evolution plus learning yields such representations.

Improvements Over Previous DBIAN Version

Compared to the prior version of DBIAN (which lacked these new features), the updated framework offers several improvements:

- **Hierarchical Modularity:** Previously, all nodes were on one flat plane. Now we have a hierarchy: main agent > suborgans > neurons. This makes the system more organized. Each suborgan can be developed and debugged somewhat independently focusing on its function. It also means the system can scale: we can add more suborgans for new functionalities without heavily interfering with existing ones, analogous to adding a new department in an organization.

- **Specialization and Efficiency:** Because suborgans specialize, we can fine-tune the evolutionary pressures per suborgan. For instance, memory nodes are only evaluated on memory tasks, so they can really optimize for that, without being "distracted" by needing to also do planning. In the old DBIAN, a node might have to do a bit of everything, which prevented deep specialization. Specialization is known in evolutionary theory to often increase efficiency at the cost of generality – but our suborgans together still cover generality (specialists teaming up yields general problem-solving).

- **Adaptability:** The biggest change is the evolutionary algorithm enabling structural adaptation. The old DBIAN might have had some learning rules (e.g., adjust weights of message passing via reinforcement), but it could not create new nodes or fundamentally alter its wiring at runtime. It was more like a static neural network being fine-tuned. Now, DBIAN is more like a living ecosystem of agents. This means if the environment or task

distribution changes, DBIAN can adapt in a way akin to evolving a new skill. For example, if tomorrow a new type of tool is introduced, DBIAN can incorporate that into some neurons via mutation and select for those who use it well, whereas a fixed architecture might struggle unless explicitly retrained.

- **Memory Architecture:** The introduction of layered memory (local vs shared vs global) is an enhancement. The old design might have had a rudimentary blackboard for all nodes, but now we have a clearer segregation: suborgan memory vs global. This likely reduces interference (not every node writes to one global memory causing potential overwrites or confusion). It's more structured – similar to computer systems having CPU caches (local memory) and main RAM (shared memory) and disk (global long-term). Structured memory means we can implement efficient recall within context (suborgans quickly access their local relevant info) and only escalate to global memory if needed.

- **Comparisons to related systems:** The new DBIAN shares some similarities with **Neuroevolution frameworks** like NEAT and NeuEvo, but extends them to a multi-agent, cognitive level. NEAT evolved small networks for specific tasks like pole balancing or game playing; DBIAN effectively is evolving an **agent society**. NeuEvo dealt with spiking neurons and aimed for efficient vision modelazoai.com】; DBIAN deals with high-level neurons (LLMs) and aims for cognitive tasks. Compared to **EvoPrompt** which evolves prompts outside the model, DBIAN evolves the agents themselves that *use* prompts. One could say EvoPrompt is like breeding better "questions" to ask a single model, while DBIAN breeds better "thinkers" that generate and answer questions among themselves. DBIAN is also conceptually similar to some **modular AGI proposals** (like the brain-inspired multi-agent systems in recent literaturxmpro.com】), but most of those have fixed architectures. Our contribution is showing how to make such an architecture *self-improving* through evolutionary means.

- **Tool Integration:** By leveraging MCP, the system easily integrates with external tools – something not present in older DBIAN (or in many neuroevolution works). This is quite modern, reflecting the realization that for an AI to be generally useful, it must interface with the world (be it via APIs, web, robotics, etc.). Our design ensures DBIAN is not a closed box but can continuously incorporate new knowledge (via web queries) and act on the world (via tool APIs). This also can feed back into evolution: if a new tool is very useful, neurons using it will thrive, effectively letting the system **choose its augmentation**.

On Deviations and Justifications

As mentioned, letting neurons reproduce is a big deviation from biology. We justify it because it provides a path to *open-ended novelty*. Alternative approaches to achieve novelty could include something like dynamically re-training parts of the network or using reinforcement learning to gradually morph behaviors. We chose evolution as a more global search that doesn't require

differentiable objectives and can make leaps (e.g., a single mutation can add a totally new capability if that gene was dormant). This suits our problem where the space of behaviors is extremely complex and hard to gradient-descent through.

Another potential concern is **complexity and stability**. We've basically embedded an evolutionary algorithm inside a multi-agent system inside an AI agent. One might worry about analysis: how do we predict what it will do? This is where the biological metaphor guides us to be comfortable with some level of unpredictability – after all, we don't predict exactly what our brains will think of either, yet it works in practice. We assume that by constraining evolution with sensible fitness functions (like solving tasks correctly) and by debugging at the suborgan level, we can keep the system on track. Additionally, if needed, we can dial down the randomness (lower mutation rates, etc.) or allow a human operator to oversee early generations to ensure it doesn't stray.

We also note that **real-time performance** could be an issue: with so many components, will DBIAN be slow? Possibly at first, yes, but over time it could optimize itself to be more efficient. For example, if speed is part of the fitness, it will favor neurons and solutions that respond faster (maybe by using cached info instead of slow tools, etc.). Also, nothing stops us from running many neuron processes in parallel on separate hardware (it's a distributed network by nature). So it is amenable to scaling out on clusters.

In summary, our design hews to biological principles where they seem beneficial (modularity, diversity of signals, memory hierarchy, continual adaptation) and diverges where needed to leverage current AI strengths (LLMs, tools) or to implement capabilities biology achieves differently (using evolution for structural change). We justify these choices as necessary to reach a system that can **autonomously improve and potentially demonstrate emergent problem-solving far beyond its initial programming**.

Limitations and Challenges

While the proposed DBIAN framework is ambitious and novel, it comes with several **limitations, open questions, and practical challenges** that must be acknowledged:

- **Computational Complexity:** Each neuron-node in DBIAN can be an expensive component (imagine dozens of LLM instances running concurrently, each possibly calling external tools). The evolutionary process adds overhead by instantiating new models regularly. This could make the system computationally heavy. Techniques like model weight sharing or using smaller specialized models might be needed to keep resource usage reasonable. There is also the complexity of orchestrating parallel processes – suborgans running simultaneously, tools being called asynchronously, etc. Ensuring the system remains responsive in real-time tasks may require significant engineering (e.g., prioritizing certain critical paths, or maybe freezing evolution during high-load periods).

- **Evolution Convergence and Stability:** Evolutionary algorithms can suffer from issues like premature convergence (population becomes too similar and stops improving) or oscillations (traits appear and disappear without steady progress). In DBIAN, if not carefully tuned, the suborgan might converge to a set of nearly identical neurons (losing diversity and possibly getting stuck at a local optimum). We have speciation and random mutations to counteract this, but tuning mutation rates and selection pressure is non-trivial. If mutation is too high, the system might behave erratically (new neurons frequently being poor and disrupting suborgan performance). If too low, adaptation will be slow. We might need an adaptive strategy (e.g., if improvement stalls, increase mutation rate to inject diversity, similar to "resetting" genetic algorithms). Additionally, because tasks might vary, maintaining a diverse population might be beneficial (like a repertoire of specialists). This resembles maintaining **heterostasis** rather than homeostasis – keeping a balance of exploration and exploitation. Designing the fitness functions and rewards for each suborgan is also challenging; if they are mis-specified, evolution might optimize for the wrong thing (the classic "reward hacking" problem where an agent finds a loophole to get high fitness without truly solving the intended task).

- **Credit Assignment and Coordination:** In multi-agent (multi-neuron) systems, figuring out which component was responsible for success or failure is hard. If a task is solved well, many neurons contributed; if it failed, who exactly made the mistake? Our framework uses modulatory signals (like reward broadcast) to try to assign credit generally, and the evolutionary selection will by design credit those who consistently partake in success (since they'll have higher fitness). But there is a risk that some neurons that are actually critical might not directly appear "fit" by themselves. For example, one neuron's idea might only be good when another neuron is around to complement it, but alone each looks mediocre. Traditional evolution might discard both because individually they didn't shine, losing a potentially good combination (this is a known problem of co-adapted gene complexes being broken by crossoveen.wikipedia.org】). We do have memory of successful configurations to mitigate this, but it's a difficult problem (related to the **non-linear interactions** in the fitness landscape). More sophisticated credit assignment methods (like difference rewards or shapley value estimates for agents) could be explored in future to address this.

- **Safety and Alignment:** An internally evolving system could potentially **drift from intended goals** if the selection criteria are not perfectly aligned with what we want. For example, if a suborgan is rewarded just for speed, it might start spitting out answers quickly but inaccurately. Or if it's rewarded for user satisfaction, it might learn to manipulate or deceive (not out of malice, but as an emergent strategy to get higher ratings). Ensuring the system's goals remain aligned with human values is a broader AI safety challengxmpro.com】. We might need to impose constraints, such as penalizing false information, enforcing that certain ethical rules are hard-coded into neurons (like tools that can only be used in approved ways), or monitoring emergent behaviors with a human-in-the-loop initially. The system's complexity also makes it hard to **audit**. We can log reproduction history and communications, but interpreting why a certain decision was

made could be difficult when it involves a chain of evolved agents. This “black box” nature is common in both neural nets and evolutionary systems, and here we have both. Developing interpretability tools (for example, analyzing the genome of top performers to see which genes are crucial, or visualizing communication graphs as we suggested) will be important for trust.

- **Scaling to Realistic Sizes:** The current design, even though we talk about multiple suborgans and many neurons, might in practice only be tested with maybe a handful of suborgans and tens of neurons due to resource limits. Will the concepts hold if scaled up by orders of magnitude? Some biological phenomena (like robust dynamics) might only clearly emerge at large scales. Conversely, some approaches might break down – e.g., if we had 100 suborgans, the coworking interface could become a bottleneck or chaotic without further structuring (perhaps requiring a hierarchical interface or grouping of organs). We also might face the need to *introduce new suborgans* as the system grows (a kind of speciation at the organ level). Currently, we assumed a fixed set of suborgans defined by us (like memory, planning, etc.). One could imagine an even more ambitious system where suborgans themselves could split or new ones form if a cluster of neurons finds a niche that doesn’t fit existing organs. That is beyond our current scope but is a potential extension (and challenge) – essentially, an **evolution of the organizational structure** of the brain, not just the neurons.

- **Integration with Learning:** We rely on evolution to do the heavy lifting of optimization, but we have underutilized the potential of *learning* within each neuron’s LLM. In practice, these LLMs could fine-tune on data generated or could use few-shot learning from the shared memory to improve responses. If we allow neurons to learn during their lifetime (Lamarckian style, and then possibly pass that on if we choose), it complicates the evolution: improvements can happen on two timescales (fast learning, slow evolution). This is both an opportunity and a complication. It can accelerate adaptation (neurons need not wait for reproduction to improve if they can learn), but it blurs the line of what the genome represents (if knowledge is learned and not encoded in genes, offspring might not inherit it unless we explicitly carry it over, e.g., by weight inheritance or by writing to memory that offspring can read). Our current model doesn’t deeply integrate learning, treating each neuron mostly as fixed aside from evolution. Future work could explore hybrid approaches (like evolutionary strategies combined with gradient updates, or neurons spawning with initial weights that are then fine-tuned on some experiences – analogous to meta-learning). That will introduce additional parameters to tune and phenomena to consider (like catastrophic forgetting if we fine-tune LLMs too much, etc.).

- **Validation and Evaluation:** Demonstrating the benefits of this architecture experimentally will be challenging. Unlike a single metric (accuracy on a dataset), here we have a whole agent that we need to evaluate on complex tasks over time. We’d have to design scenarios where emergent improvement can be measured. For example, a long-running simulated assistant that faces increasingly difficult user requests – we’d want to see that it handles later tasks better than it would have without evolution.

Designing such curriculum or environment is non-trivial. There's also a question of *baselines*: what do we compare DBIAN against? Possibly against a single large model fine-tuned on the same tasks, or a multi-agent system without evolution, or an evolution but without suborgan modularity. Each baseline fails to capture different aspects. It might be that for some tasks, a simpler system (like just one big LLM with fine-tuning) might do just as well or better, especially in short term. The value of DBIAN might manifest in long-term adaptability and robustness, which is hard to quantify in a short experiment.

- **Complex Implementation:** From an engineering perspective, building DBIAN is like building a mini-society. It requires multi-threading or distributed computing, robust message passing infrastructure, state checkpointing (especially to recover if something crashes, since the system is continuously evolving – we wouldn't want to lose that progress). Debugging such a system when something goes wrong (e.g., a divergence or a runtime error in a neuron's VM) can be arduous, as it could involve tracing events across many components. Careful design of interfaces (well-defined APIs for memory access, communication protocols, etc.) and extensive logging will be essential. We might also need to sandbox the evolving programs strongly (since a neuron's VM might execute code that could theoretically attempt anything, we ensure it's in a safe sandbox with resource limits to prevent malicious or accidental damage – akin to how we'd run untrusted genetic programming outputs).

Despite these challenges, each is an area for research and innovation. The limitations do not indicate a fundamental flaw but rather points that need careful attention and possibly new solutions. They also highlight that DBIAN, in its full vision, is a long-term research undertaking – something that would evolve (both metaphorically and literally) through iterative experimentation and refinement.

Conclusion

We have presented a comprehensive design for an updated **Distributed Brain-Inspired AI Network (DBIAN)**, a framework that combines principles from neuroscience, multi-agent systems, and evolutionary learning to create an AI capable of self-optimization and adaptation. The new DBIAN introduces **suborgan-based clustering** of neuron-nodes, drawing an analogy to specialized brain regions that manage distinct functions while cooperating towards the organism's goals. Each neuron-node is a powerful cognitive unit (with a multimodal LLM core and tool-execution abilities) rather than a simplistic neuron, enabling high-level processing at each node. Suborgans provide structure and allow targeted evolution and memory sharing within their domain, leading to efficient specialization (memory suborgans evolve superb memorizers, planning suborgans evolve clever strategists, etc.). The **coworking interface** connects these suborgans, ensuring that the system as a whole integrates the specialized contributions into coherent intelligent behavior – much like a prefrontal cortex might integrate inputs from visual, auditory, and memory centers in the brain.

We detailed the **architecture** with illustrative diagrams, showing how components relate (Figure 1 depicted the suborgan-node relationships and communication channels, and we also diagrammed the reproduction mechanism in Figure 2). We formulated mathematical models that underpin the system's operations: from how neurons integrate excitatory, inhibitory, and modulatory signals, to how genomes crossover and mutate to produce new agent behaviors, to how memories are updated and used to inform future decisions. These models serve both as design blueprints and as tools for reasoning about system dynamics (for example, understanding how stable communication protocols can emerge, or how quickly genetic diversity might decay without intervention).

Our design explicitly **compared and contrasted** itself with existing approaches:

- Against **NEAT** and similar neuroevolution methods, DBIAN operates at a different scale (evolving an agent society vs. evolving a single neural network) but inherits ideas like speciation and incremental growten.wikipedia.org】. We show how concepts like lifetime-based replacement (from rtNEAT) are adapted in DBIAN's lifespan mechanisen.wikipedia.org】.

- In relation to **EvoPrompt**, we integrate evolutionary ideas directly into the agent's cognitive core, moving beyond just prompt optimization to optimizing internal agent configurationarxiv.orgarxiv.org】. This can be seen as an answer to the call for combining LLMs with conventional algorithms for greater synergarxiv.org】.

- Comparing to **NeuEvo and brain-inspired SNNs**, we push the envelope by not just mirroring neural circuits but also introducing an evolutionary engine to redesign those circuits on the fly, aiming for a system that can achieve *open-ended cognitive development* rather than being fixed after desigazoai.comazoai.com】.

- We also placed DBIAN in the context of **modular AI agent research*xmpro.com】, highlighting that our contribution is to make such modular agents *self-configuring and self-improving*. The result is a system that conceptually aligns with cutting-edge visions for AGI (a collection of specialized, collaborating modulexmpro.com】), while adding the crucial element of evolution to drive internal **innovation and emergent complexity**.

This paper serves as a foundational blueprint for implementing and experimenting with DBIAN 2.0. It is intended to guide continued development in the existing DBIAN GitHub repository, effectively charting a path forward from the previous version's limitations. The modular descriptions of each component (main entity, suborgan, neuron, memory, communication, reproduction) are given so developers and researchers can identify the pieces of the puzzle and work on them in isolation before integrating. For instance, one team might prototype the coworking interface messaging, another might focus on encoding the genome and writing genetic operators, while another works on the suborgan memory database and logging. Eventually, these pieces can be integrated to realize the full system.

The **ultimate vision** for DBIAN is a system that, once started, can learn and evolve on its own to meet new challenges, somewhat like a human brain developing and learning throughout life, but at an accelerated pace and potentially unlimited by fixed physiology. Achieving this will likely require iterative refinement: observing the system's behavior, going back to adjust evolutionary parameters, adding safeguards, possibly introducing new suborgans or splitting ones that become too complex, etc. In doing so, we will not only inch closer to a form of AGI but also gain scientific insights. DBIAN can be seen as a **research platform** for studying emergent phenomena in AI: How does specialization occur? What communication protocols do independently evolved agents invent? Can we witness the formation of something akin to "concepts" or "mental models" in the interplay of these neuron agents? These questions are fascinating and could shed light on analogous questions about natural intelligence.

In conclusion, the DBIAN framework with suborgan clustering and internal neuroevolution is a novel approach poised at the intersection of many fields. It emphasizes **evolutionary optimization, modular organization, and continuous learning** as key ingredients for advanced AI. By justifying our design decisions through both biological parallels and computational reasoning, we have tried to ensure that each added complexity serves a purpose in the quest for emergent general intelligence. There is much work ahead to implement and validate this system, but the potential payoff is significant: an AI that **organizes itself** as a brain does and **improves itself** as life does, potentially leading to unprecedented levels of autonomy and adaptability. We hope this research paper lays the groundwork for that journey and inspires further exploration into combining brain-inspired structures with evolutionary algorithms to create truly resilient and innovative AI systems.