The Corelian Architecture

Corely's cognitive documentation

Harith Marzuki

Contents

Forewords

This paper is a documentation of a completed personal project to build an AI robot named **Corely.** This self-initiated project started as a small coding experiment in attempting to find a new way to build an AI. It then expanded into an exploration of understanding human cognition and trying to break it down into computer algorithms. The result of this experiment thus produced a computer program that mimics the human brain as a centre of cognition.

This project underwent two different names, *Project Corely* and *Project Nano Corely*. Each name reflects different phases of the project development.

Project Corely was focused on laying down the foundation and designing the main architecture. This is where most of the system's mechanisms were created and tested. It primarily ran as a program running on a laptop while connected to a web page opened on a phone, which serves as Corely's head.

Project Nano Corely was a step up to bring Corely out of the laptop and onto her own body. This pivot was driven by the limitation of Project Corely, where she didn't have the means to move herself. As such, this phase of the project was primarily about building the robot body and integrating the existing system onto a Raspberry Pi Zero 2. This changed Corely from simply existing as a program to having an autonomous body.

This documentation is thus based on the system created from Project Nano Corely, as it is considered the complete iteration of Corely's system. Regardless, both projects have been uploaded to GitHub separately for public reference:

- Project Corely: https://github.com/HarithMarzuki/Project-Corely
- Project Nano Corely: https://github.com/HarithMarzuki/Project-Nano-Corely

Although this is a project documentation, its contents could serve as a source of inspiration for innovation in AI, either as a whole or by parts. The author believes that the pursuit of AI should be a collaborative effort among humans, and that any benefits arising from it should be shared equally.

This project was not affiliated with any government, business or institution, and has not received funding from public investors.

Introduction

Corely is an AI by traditional definition. However, the way she was created and the purpose of her system are unlike most AI or machine learning models found in mainstream research and industrial use. Most AI systems are built for the purpose of task completion, with the end results and benchmark scores being their measure of effectiveness. This top-down approach is excellent for creating a helpful assistant for day-to-day business, but for achieving an Artificial General Intelligence that could incentivise, initiate and lead tasks on its own end-to-end, this top-down approach frequently reaches its limit. This limit is primarily due to the situational awareness gap that is present in top-down environments, where goal completion is prioritised over all else, often leading to mistakes in the form of hallucination or bias.

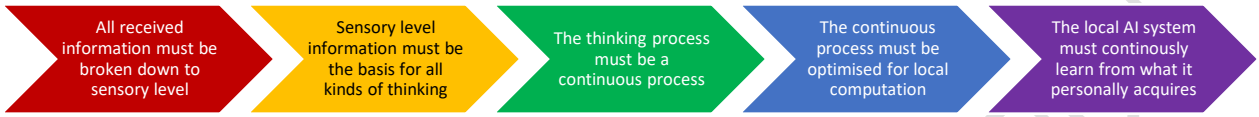
Corely is fundamentally the opposite of the top-down approach AIs. She is created from the bottom up, with the design priority being placed on her cognition and thinking processes rather than the end result she produces. There is no training data or even a training phase, for her system continuously learns from streaming inputs, which are filtered before it was consolidated into her expanding memory.

It must be clarified that because of this bottom-up approach, she is not guaranteed to be able to perform a task or follow instructions as given. This can be seen as useless from the industry's perspective that would normally want an immediate use case, yet this experimentation could be paving a way to something that is indescribable for us today but invaluable in the future.

Core principles

The Corelian Architecture is the name given to Corely's AI system. To understand how this AI system works, it is important to first grasp the core principles that shaped its development.

The Five Principles of the Corelian Architecture



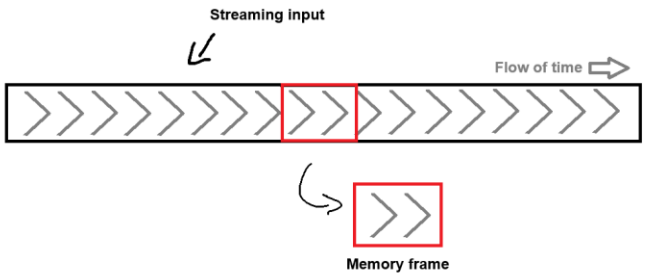
- All received information must be broken down to the sensory level.
    - Humans are capable of learning a wide variety of knowledge, yet all of it is received through the basic senses.
    - This is commonly associated with gaining experience through interaction, especially among children. Outside of humans, this form of learning is also seen with intelligent animals.
    - However, this concept actually applies beyond basic physical interaction. Learning academic knowledge still requires at least sight and hearing senses to be available in order to make the learning material accessible for the learner. For the visually impaired, this sensory requirement is substituted with their touch senses instead via braille.
    - Nonetheless, this shows that in order to have general intelligence, the senses must first be general in nature. This allows chaotic information from the real world to be generalised into standard forms through sensory filtration.
- Sensory level information must be the basis for all kinds of thinking.
    - Because the information has been broken down by the senses, it has to be reconstructed inside the brain to become knowledge. The most basic way to do this is by associating information across different senses together by their simultaneous occurrence. This is the basis for creating a mental simulation of the world.
    - However, knowledge reconstruction can be done in other ways as well. The sensory level information can be reorganised and grouped to simulate events or knowledge that aren't in the real world. This is the basis for imaginative and creative thinking.
    - Reconstructed knowledge in the brain can itself be synthesised into smaller information that isn't available from the senses and be used to create other knowledge. This is where abstractive and deductive thinking comes to form.
    - Regardless of which form of thinking is performed, it all uses the same resources that originated from the senses. This keeps the emerging intelligence grounded in reality while allowing for mental exploration.
- The thinking process must be a continuous process.
    - The human brain continuously processes information coming from the external senses and the internal thought without ever fully stopping. This is a key feature

in order to have situational awareness, as it allows the brain to maintain a present state while keeping tabs on multiple different pieces of information at once.
    - Situational awareness also allows the brain to pick up subtle information and changes which could be dismissed at first but are critical for forming thoughts later.
    - Additionally, it can create bridges across different unrelated thoughts as new information streams into the brain, allowing for inspiration and intuition to form.
    - This shows that situationally aware intelligence not only needs to keep present information on hand, but also allow thoughts to process that information even when it is not immediately needed.
- The continuous process must be optimised for local computation.
    - Supporting continuous processing requires a robust AI system that could handle both streaming information from the real world and internal computation simultaneously. Any latency along the process pipeline can greatly slow down the AI system's responsiveness as information adds up.
    - Reducing latency involves many parts, either by optimising the computation or reducing signal lag.
    - Local computation is the most reliable way of reducing signal lag as it ensures lossless data transmission at maximum speed. However, it regularly comes with limited computational power, which lowers the AI system's maximum performance.
    - This lower performance issue can be overcome by separating the AI system into multiple lightweight components that run asynchronously.
    - In this asynchronous environment, the primary component would continue working with they have first while waiting for any requested information to be given to it by the secondary component. This ensures that the master component is always ready to process new information while the slave component handles slower processes.
- The local AI system must continuously learn from what it personally acquires.
    - Apart from knowing what information it has, the AI system needs to also know when it got the information. This is to help establish the information chronology, which is important for process understanding.
    - This form of understanding opens up more capabilities, as it allows the AI system to create plans based on learned experience, update its knowledge without overwriting the old knowledge and form its own rationale.
    - Having the AI learn from its personal experience would also reinforce its situational awareness and adapt to the local environment.

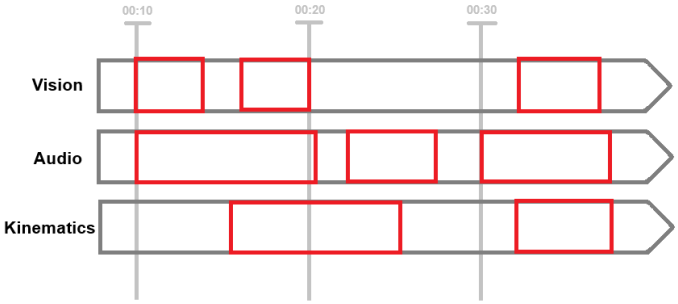
General Mechanisms

The architecture serves as a general framework for managing saved memories, internal states and decision-making processes on a unified platform. It acts as **a cognitive engine** for the AI system to cycle data from both the real world and internal memory sustainably for an extended period of time. The engine has to have the following mechanisms for it to work:

- Memory framing



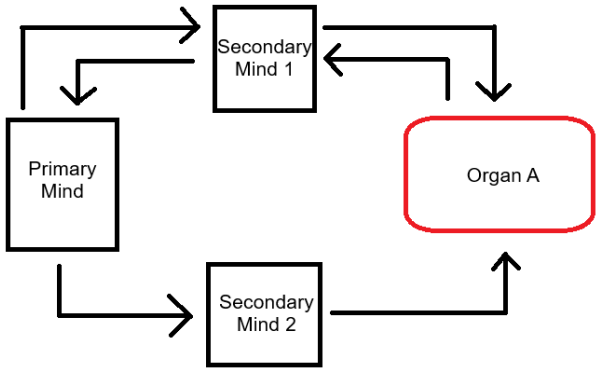
  - Not everything from streaming input is worth saving. Most of them are either noise or redundant to the memory.
  - The system should be able to capture a frame of memory from the streaming input as a basic unit of information.
  - Each type of sensor can have its own framing strategy, depending on its suitability for the input data.
  - A memory frame would contain a snapshot of the input data as well as attributes that describe its features, its relation to other frames and the status of the system.
  - Memory frames are the medium on which the cognitive engine runs.
- Chronological track



  - Chronological tracks are where memory frames are saved sequentially. Memory frames are placed onto different tracks based on their sensor type.
  - The tracks run parallel to one another, meaning memory frames from different sensors that were framed in the same time period would be adjacent to one another on their respective tracks.

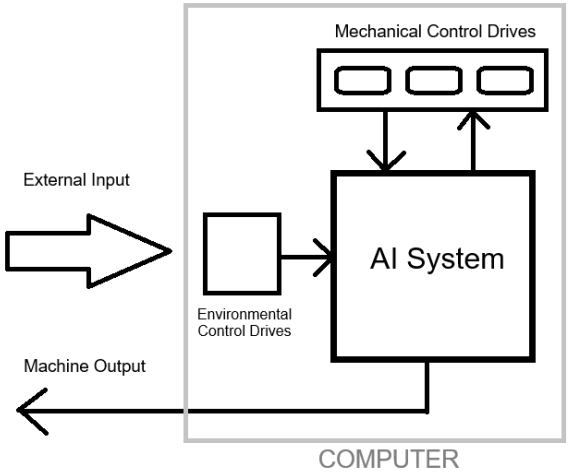
- The memory frames would store attributes referencing their neighbour, either from the same track (previous and next frames), or from different tracks (time-adjacent memory frames)
- The chronological tracks serve as a navigable medium for individual components to fetch saved memories efficiently.

- Cognitive components and pipelines



- The cognitive engine contains multiple components that are responsible for handling different functionality of the system. The components are connected via data pipelines that dictate the flow of memory frames between them.
- Components are divided into two types: **Minds** and **Organs.**
- **Minds** are active threads that perform executive tasks for the system, allowing the cognitive engine to operate. They run autonomously and push memory frames along the data pipelines.
- **Organs** are static codes, files, or variables that support the operations of the system. They are passive resources that do not perform tasks on their own but readily await the active minds to use them.

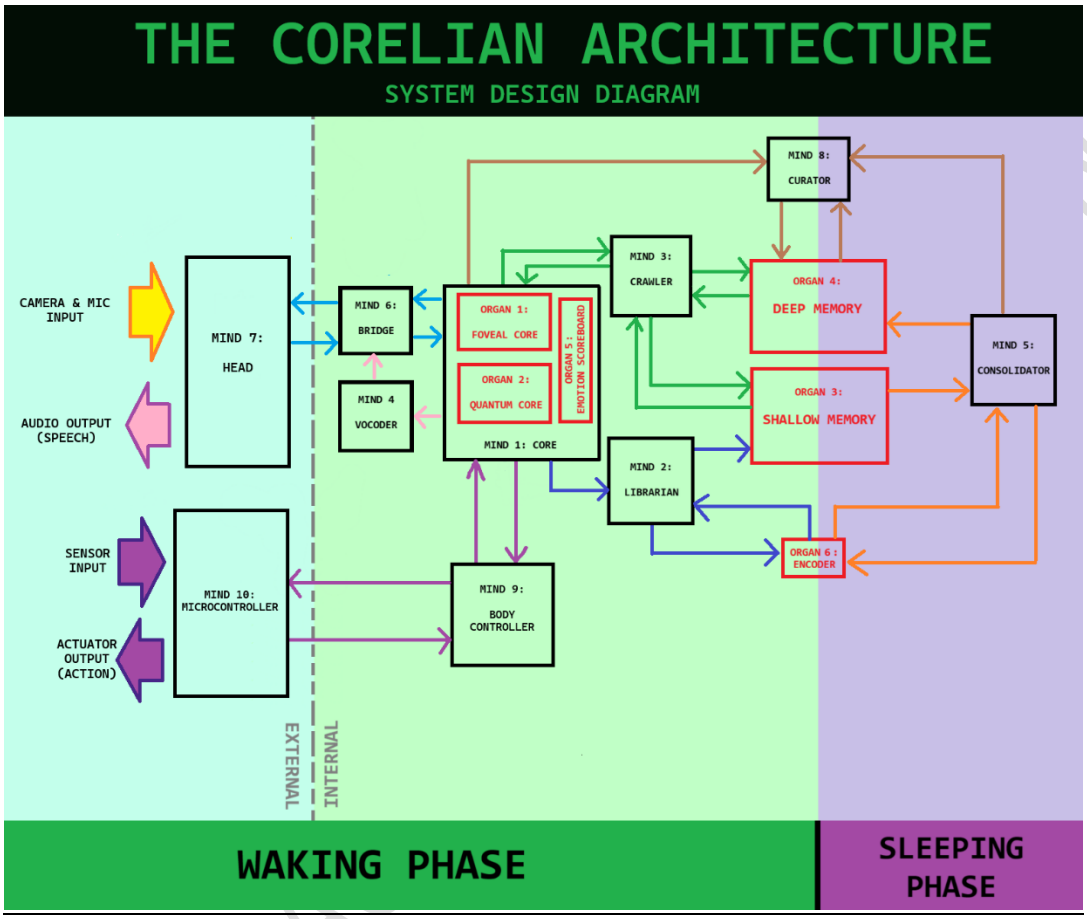
- Control drives



  - Although the minds of the cognitive engine are autonomous operators of the system, they require controllers that regulate their actions.
  - Control drives are variables that track the present status of the system and provide it to the minds to direct the next course of action. This status can be derived from either physical sensor data or internal system values.
  - This creates a feedback loop as the minds' executive tasks affect the status, and the status affects the minds.
  - Two types of control drives are present simultaneously inside the system: **Mechanical** and **Environmental.**
  - **Mechanical** control drives are simple variables that keep track of values that are secondary by-products of the minds' action. Their behaviour should be **deterministic** in theory, as everything is sourced internally.
  - **Environmental** control drives are random values derived from **non-deterministic** sources. This value can be obtained either by processing sensor inputs through a non-deterministic transformer, or from external hardware that can produce true random values. *Pseudo-randomisers are not recommended as they are inherently deterministic.*
  - The two types of drives are there to balance the AI system's behaviour between being completely mechanical and completely random.

System Design

The following diagram shows the system design of Corely's system. Black boxes indicate active components, while red boxes indicate passive components. Additionally, the arrows show the flow of data and memory frames, coloured based on their group.



Description of each component:

- Mind Components

No.	Name	Function
1	Core	• The primary mind coordinating the operations between different minds and organs in the waking phase.
2	Librarian	• Handles the encoding and writing process for new memory frames entering the shallow memory.
3	Crawler	• Searches memory frames in deep and shallow memory requested by Core.
4	Vocoder	• Converts audio memory frames into speaking voice.
5	Consolidator	• Consolidates memories from shallow memory to deep memory in the sleeping phase.

6	Bridge	• Manages wireless connection between Core and Head.
7	Head	• Streams camera & mic input, plays speech, and visualises facial expression.
8	Curator	• Creates chain-of-thought by linking multiple memory frames into a ‘story’.
9	Body Controller	• Encodes action commands to the microcontroller and decodes sensor data from the microcontroller.
10	Microcontroller	• Executes action commands by activating actuators and encodes sensor data.

- Organ Components

No.	Name	Function
1	Foveal Core	• Special component that captures the centre of attention from visual input inside a looping multi-layered diffusing array, acting as temporal memory space for vision.
2	Quantum Core	• Environmental control drive that derives its random values from the foveal core.
3	Shallow Memory	• Memory bank for newly written memory frames which are unconsolidated.
4	Deep Memory	• Memory bank for consolidated memory frames with cluster groupings.
5	Emotion Scoreboard	• Tracks the emotion values which influence memory searching.
6	Encoder	• Algorithm for profiling raw memory frames with encoded parameters used in many operations.

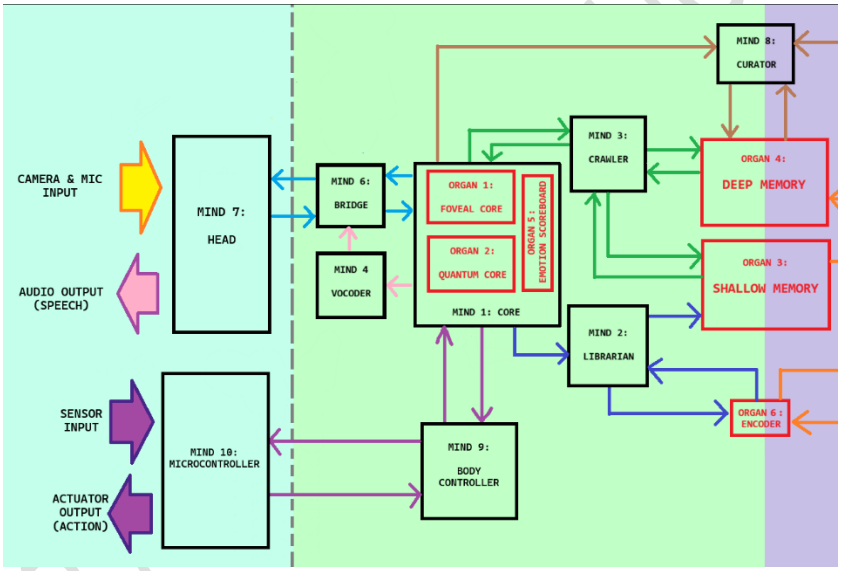
System Operation

The following section contains the implementation of the Corelian Architecture found inside Corely. However, it has to be stated that these prototypical implementations may not be the best approaches, and there is room for improvement for each aspect using more advanced techniques. Nonetheless, the following parts reflect how the system is currently, and may serve as inspiration for future innovations.

Part 1: Operational Phases

Corely's system operation is divided into two phases: the **waking phase** and the **sleeping phase.** This division is created to protect the memory banks from catastrophic failure caused by the conflict between the Crawler's and Librarian's rapid read-write operations and the Consolidator's slow restructuring operations. This allows the memory bank to remain optimal for quick searching without hindering the system's active operation.

- The Waking Phase



i. This is the primary operational phase, where the system is active to interact with the external world and its internal memory.

ii. Upon the Core's activation, it will activate Librarian, Crawler, Vocoder, Bridge, Curator and Body Controller alongside it.

iii. The Bridge will establish a connection with the Head, while the Body Controller will establish a connection with the Microcontroller.

iv. After the connections are active, the camera and microphone will stream their raw input to the core via the Bridge's connection.

v. The raw inputs are then processed via separate filtering pipelines inside the core to produce memory frames. *(Specific details on the filtering pipelines can be found in the next part)*

vi. After obtaining a fresh memory frame, the Core would decide what it wants to do with it using control drives.

vii. Control drives have weighted probability and are selected randomly using the Quantum core's random value.

viii. Some control drives would lead to the memory frame being appended into shallow memory by the Librarian, while other control drives would lead to the Crawler fetching a similar memory frame to compare them.

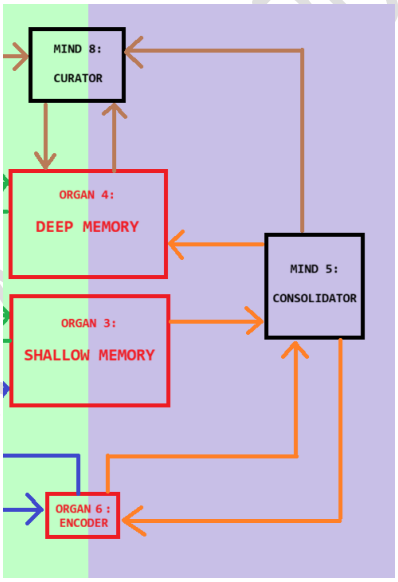
ix. If the fetched memory frame has 'stories', the Core would have a chance to select one story and consecutively fetch a series of linked memory frames. *(More details can be found in the 'Curator & Stories' part)*

x. Some fetched audio or kinematic memory frames would also be used for output based on various conditions. The vocoder would use the audio memory frame as speech, while Body Controller would use the kinematic memory frame as action commands.

xi. While the main process is running, the Curator would perform its story-making process in the background *(More details can be found in the 'Curator & Stories' part)*

xii. This process will continually loop until the core is deactivated, where all active minds would also be deactivated.

- The Sleeping Phase



i. This is the secondary operational phase, where the Consolidator becomes active to consolidate new memory frames.

ii. The Consolidator's process starts by scanning through the deep memory banks for pruning memory frames that have weights below a certain threshold.

iii. Additionally, it would also clean up all the memory clusters that exist in the deep memory bank.

iv. Next, memory frames from the shallow memory bank are transferred into the deep memory, and an unsupervised clustering process is done to cluster memories together.

v. After the clustering process, the Consolidator would activate the Curator to create stories using any of the saved memory frames. The number of stories to be created here is based on the square root of the number of memory frames inside the deep memory.

vi. After the Curator's process is done, the Consolidator would shut down, and the system is ready to be awakened again.

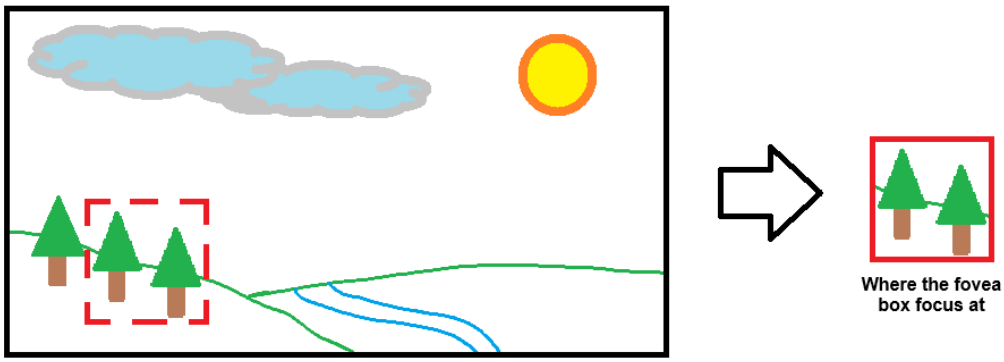
Part 2: Sensory Pipelines

Vision, audio and kinematics are the 3 basic senses needed for an embodied AI system. This is because they have unique Input/Output pipeline that cannot be adapted directly to the other basic senses, as shown in the table below.

Types of senses	Input	Output
Vision	Sight	X
Audio	Hearing	Speech
Kinematics	X	Action

Additionally, each sense is found to exist fundamentally in a different dimension. For example, visual memory can be stored as 2D static images, but audio memory has to be stored as a continuous sample. On the other hand, both visual and audio processing pipelines have to accommodate external input, while the kinematic pipeline has no source for external input in the first place. As such, each sense requires its own processing method with vastly different approaches and goals.

- Visual Pipeline – Fovea Box Stability.



i. When raw visual input reaches the Core, the image is copied onto a visual canvas.
ii. Inside the visual canvas, there is a fovea box that gradually moves across the space toward the most visually active region. The visual activity is based on the difference between the previous and the current input frame.
iii. When the fovea stabilises over a region, the current input frame will be captured as a visual memory frame.
iv. After a fresh visual memory frame is obtained, the Core would decide what it wants to do with it using cognitive control drives.
v. There are 3 cognitive drives: Save, Predict & Wander. The drives have weighted probability and are selected randomly using the Quantum core's random value.
vi. The drives' probability weights fluctuate dynamically based on the different conditions:
    - Save: Increases when the visual motion is high.
    - Predict: Increases when the previous prediction is correct.
    - Wander: Increases when visual motion is low or the audio input is quiet.

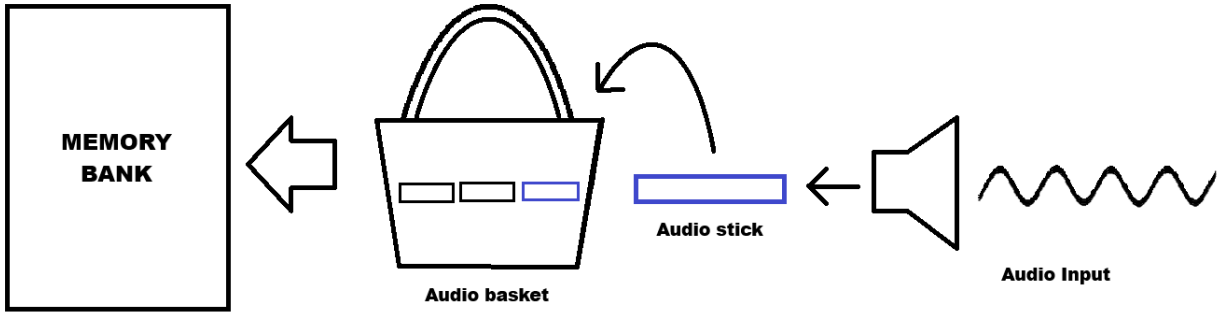
vii. If the ‘save’ drive is selected, the memory frame would be sent to the Librarian’s queue, where the Librarian would then profile the memory frame using the Encoder and append it to the shallow memory on the visual chronological track. The sensor values from the microcontroller are saved alongside the visual memory as attributes.

viii. If the ‘predict’ drive is selected, the Core would call the Crawler to search for similar memory frames in either the shallow or deep memory and fetch the following memory frame to use as a prediction. If it succeeds, then the fetched memory would be rewarded. If it fails, the ‘save’ drive would be activated immediately.

ix. If the ‘wander’ drive is selected, the Core would call the Crawler to search for random memory frames in either memory bank and fetch them to be immediately rewarded.

x. Any visual memory frame fetched by the Crawler would be diffused with the real vision on the visual canvas.

- Audio Pipeline – Stick and Basket.



i. When raw audio input reaches the Core, each audio sample frame will have its amplitude measured; if the loudness exceeds the activation value, it will be taken in as a ‘stick’ and placed into an audio ‘basket’.

ii. If the next sample frame also has a loudness that exceeds the activation value, it will be taken as another stick and attached at the end of the previous stick inside the basket.

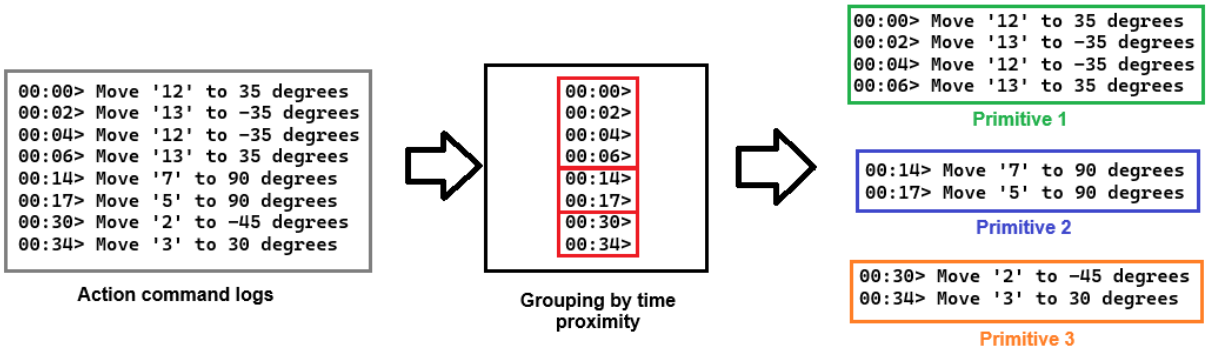
iii. This process continues until the sample frame drops below the activation value, where it will be taken as the last stick to be attached before the audio basket is packaged as an audio memory frame.

iv. The audio memory frame is then immediately sent to the Librarian, where it is appended into the shallow memory on the audio chronological track.

v. At the same time, its profile would be used by the Crawler to look for matching audio to be rewarded. If there are none, then this process will be skipped.

(This is only the pipeline for audio input. Audio outputs are highly correlated with the story mechanism, which is explained in detail in part 4)

- Kinematic Pipeline – Body Possession Protocol (BPP).



i. When the Core first activates, it will scan for the connection with the microcontroller and establish a handshake requesting the firmware version and the body schema.
ii. The Core would then check the received version number and schema with the BPP dictionary, ensuring it matches the schema it had.
iii. After the confirmation, the Core would activate the BPP driver of the correct version as the Body Controller.
iv. As the system runs, the Body Controller would be controlled by the action control drives, which are independent from the cognitive control drives.
v. There are 3 action drives: 'do nothing', 'motor primitives' and 'motor babbling'. Similar to cognitive control drive selection, these drives' probability weights dynamically shift based on the following condition:
    - Do nothing: increases up to a limit when boredom is low, decreases down to a limit when boredom is high
    - Motor primitive: share the remaining probability space with motor babbling, where a higher number of saved motor primitives increases its portion.
    - Motor babbling: share the remaining probability space with motor primitive, where a higher number of saved motor primitives decreases its portion.
vi. If 'do nothing' is selected, no action would be executed.
vii. If 'motor primitive' is selected, the Body Controller would execute a set of commands stored in the motor primitive list file, and reward its weight.
viii. If 'motor babbling' is selected, the Body Controller would randomly select an actuator with a random value set on the command's parameter.
ix. Each command sent to the microcontroller is logged into the command log, and when the time gap between two commands is greater than a threshold, all the commands with a time gap shorter than the threshold would be clustered together as a kinematic memory frame.
x. The memory frame is then sent to the Librarian to be appended into the shallow memory on the action chronological track. It would also be linked to visual and audio memory frames that it overlaps with its timestamp.

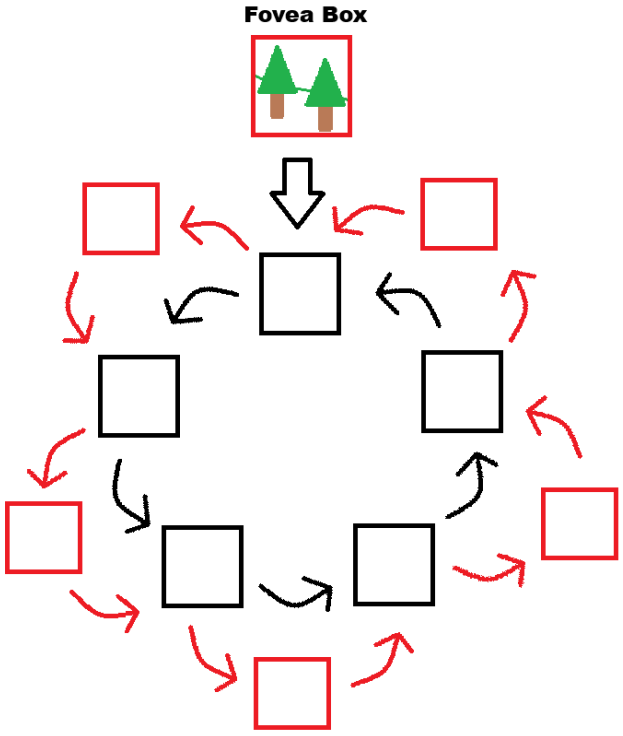
xi. When the crawler fetches visual and audio memory frames with a linked kinematic memory frame, the kinematic memory would be used as a series of commands and override the action drive selection process. Finally, the series of commands would be added to the motor primitive list file.

xii. If the fetched kinematic memory frame uses an outdated body schema, the body controller would automatically translate the memory frame onto the new schema version. This is to ensure that if the AI system ever changes from one body to another, old kinematic memories would remain usable.

Part 3: Foveal Core & Quantum Core Interaction

These two organs are central to many decision-making processes in this AI system. The Foveal Core is designed to be the temporal memory space for the visual cortex, while the Quantum Core is made to connect the Foveal Core to the rest of the system by extracting its random state.

- Foveal Core



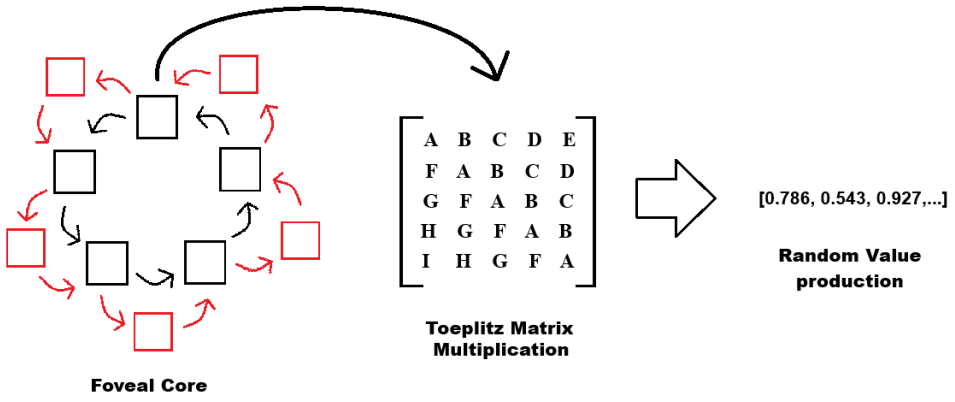
The Foveal Core is unique in a way that it preserves input in the form of lingering imprints on top of the latest input, creating a chaotic environment that is sensitive to input variation. It works as follows:

i. As the fovea box moves, all pixels that are inside the bound will have their RGB value copied to the Foveal Core.
ii. The Foveal Core has three colour channels, each containing 5 primary layers and 5 heat layers. Each layer is made of a 2D array with the same size as the foveal box.
iii. The copied RGB values are pushed onto the first primary layer, where they will diffuse onto the next layer until the last one, after which they will diffuse back to the first layer.
iv. Every few program ticks, the values of all primary layers are proportionally added to their respective heat.
v. At every primary layer's diffusion, the heat layer values would be proportionally added to the primary layer, influencing the present input diffusion.

- Quantum Core

The Quantum Core, on the other hand, comes from a personal idea to exploit quantum physics randomness inspired by Roger Penrose's hypothesis of quantum consciousness. Though this AI system was made to neither prove nor disprove Penrose's idea, the choice of wanting this level of randomness comes from the control drives selection issue, where the system's memory growth is heavily reliant on how the controls were selected. Using pseudorandomisers for this purpose would make the system deterministic; thus, the need for a non-deterministic control drives selector arises.

The key to getting pseudo-quantum randomness comes from using the camera noise, as it is originally caused by quantum physics of photons. However, directly using the camera noise has its own problem, as it is also influenced by the device's electronic interference and present environmental factors. Thus, the Quantum Core utilises the Foveal Core and a Toeplitz matrix to balance out the problems.

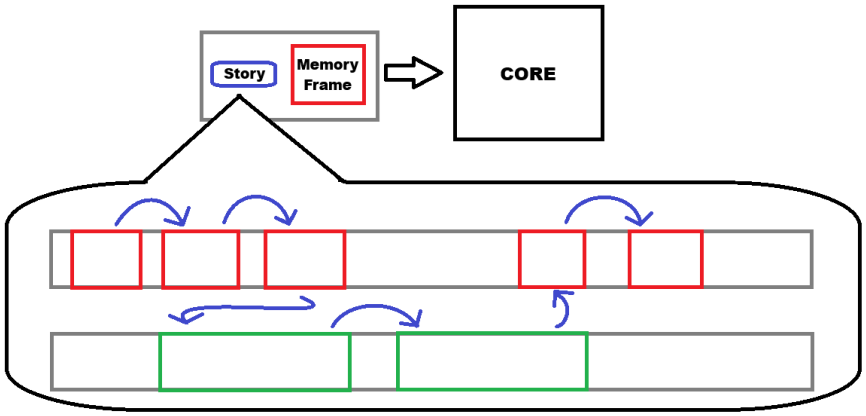


The way it works is as follows:

i. At every system tick, the Quantum Core would copy the value from the first primary layer of the Foveal Core.
ii. The primary layer is then compressed into a 1-dimensional array. The array value is then added to an internal time-series wave that updates every tick. This is to prevent similar random values being produced when the camera is static.
iii. The 1-dimensional array is scrambled using a fixed-value Toeplitz matrix, and the result is amplified to a large value before being stabilised into a range between 0 and 1 using a sine operation.
iv. The 1-dimensional array is pushed into a list, where any random calls will draw its value from here.

Using the Foveal core ensures that temporal memories influence the random value production, while remaining non-deterministic through the Toeplitz matrix scrambling. When the system made a selection using Quantum Core's random value, the Foveal Core would have its input influenced by the physical movement of the body or memory recall process, impacting the next random value production.

Part 4: Curator & Stories



Stories in the Corelian Architecture are a memory-linking mechanism created to establish basic chains of thought. The process of creating stories is done by the Curator, and it can be performed in either operational phase. They are created by connecting memory frames one after another based on their existing links and potency. The key feature of this mechanism is that the Curator isn't restricted to connecting memory frames in their successive order, but can also connect memories in reverse order, across different chronological tracks, or within the same memory cluster. This provides the AI system with quick paths to associate memory frames and discover hidden connections that would otherwise remain undetected from chronological experience alone.

- Curator in the waking phase

i. When the Core rewards a memory frame's weight, the Core would check whether its cumulative weight has passed a certain threshold. If it has passed, the memory frame would be added to the Curator's high-importance collection, which is a 'first in, first out' queue.

ii. The Curator would pick up a memory frame from the collection and place it at the front of the 'storyboard' (the Curator's workspace).

iii. It then randomly selects which mode it wants to be in, 'Reflect' or 'Imagine'.
   a. 'Reflect' mode would have the Curator trace backwards in the deep memory bank.
   b. 'Imagine' would have the Curator move forward in the deep memory bank.

iv. It would also randomly select its energy value, which limits how many memory frames it adds to the storyboard

v. It then starts the story-making process by looking up all the memory frames and clusters the first memory frame has links with, and obtains their weights.

vi. The Curator would use their weights as probability weights and randomly select a cluster or memory frame using the Quantum Core's random number.

vii. When a memory frame is selected, it would be added to the storyboard, and the Curator would work on it next. If a cluster is selected, the Curator would perform random selection on a list of the most weighted memory frames in the cluster.
viii. This process is repeated until the Curator runs out of energy, when it would then package the storyboard into a story and attach it to the first memory frame's attribute.

- Curator in the sleeping phase

i. The Consolidator would wake the Curator after the clustering process is finished.
ii. The Curator would seed a pseudorandomiser by doing a one-off Quantum Core process with frozen Foveal Core.
iii. It will calculate how many stories it should create by getting the square root of the number of saved memory frames.
iv. The remaining story-making process would remain the same as in the waking phase.

- Utilising story in the system

i. When the Core receives a memory frame from the Crawler, if the memory has stories in its attribute, it has a chance to pick up one of the stories and execute it.
ii. During story execution, the Control drives selection process would be overridden, giving the Crawler priority to fetch memory frames based on the story.
iii. If the story is an 'Imagine' story:
    a. Each visual memory fetched would be compared to the real visual input, keeping track of how many memory frames manage to match. The final score of comparison would be tallied up at the end, and if the score is positive, the story and the story-owning memory get rewarded.
    b. Additionally, when the visual or audio memory frame has a link with a kinematic memory frame, the kinematic memory would be used as action commands, depending on the current control drive.
iv. If the story is a 'Reflect' story:
    a. Visual memories fetched would not be compared to the real visual input, but the audio memories fetched would be sent to the Vocoder to be played as speech.
    b. The current emotion would be influenced by the emotion values of the fetched memories, and if the current emotion resonates with the last memory frame's emotion, the story-owning memory gets rewarded.

Limitations & Recommendations

The current implementation of the Corelian Architecture has not been rigorously tested against any relevant benchmark or empirically analysed. So far, the architecture remains experimental as its effectiveness remains unproven. Additionally, the experimentation and development of this AI system were limited by hardware, budget and expertise constraints. Thus, a more sophisticated implementation of the Corelian Architecture might exist given more research is done.

Possible areas of improvement to the Corelian Architecture would involve using advanced ML methods for some of the components, adding more features to the memory frames, integrating the architecture as part of larger AI models, or expanding the system on more powerful hardware.