

Causal Cognitive Architecture 8

Nov 1, 2024 Release

Figures

Verbal Description

Modeling Equations including Pseudocode Overview

Experimental Trial Methods and Results Data

Reproducible Python Source Code (adjacent linked GitHub)

Hooks for LLM Hybridization (linked GitHub)

**Hooks and Python Source Code for interface to PetitCat Mobile
Robotic Devices (linked GitHub)**

Howard Schneider¹

¹Sheppard Clinic North, Vaughan, Ontario, Canada

*** Correspondence:**

Howard Schneider

hschneidermd@alum.mit.edu

Verbal Description and Figures for CCA7 with transition to CCA8

1 Introduction

The Causal Cognitive Architecture (Schneider, 2023, 2024) is a brain-inspired cognitive architecture (BICA). It is hypothesized that the navigation circuits in the amniotic ancestors of mammals duplicated many times to eventually form the neocortex (Rakic, 2009; Chakraborty and Jarvis, 2015; Butler, Reiner, Karten, 2011; Fournier, Müller, Laurent, 2015; Kaas, 2019; Güntürkün, von Eugen, Packheiser, Pusch, 2021; Burmeister, 2022). Thus, the millions of neocortical minicolumns are

modeled in the Causal Cognitive Architecture as millions of spatial maps which are termed “navigation maps.”

The architecture is not a rigid replication of the mammalian brain at the lower level of the spiking neurons, nor does it attempt to behaviorally replicate the higher-level psychological properties of the mammalian brain. Instead, it considers given the postulations above, what properties and behaviors emerge from a cognitive architecture based on navigation maps. The architecture represents a more functionalist system as per Lieto (2021, 2021b), although a continuum exists between functionalist and structuralist models. Even in this paper, where an enhancement to the architecture is considered, the constraints of biology and anatomy are taken into account.

Figure 1 shows an overview of the Causal Cognitive Architecture 5 (Schneider, 2023). Figure 2 shows the operation of the architecture through a cognitive cycle—a cycle of sensory inputs processed and then an output (or null output) resulting. As noted above, the millions of neocortical minicolumns are modeled in the Causal Cognitive Architecture as millions of spatial maps which are termed navigation maps. An example of a simplified navigation map used in a simulation of the architecture is shown in Figure 3. The operation of the Causal Cognitive Architecture is given in more detail below; here a simple overview of its function is provided.

Each cognitive cycle, sensory inputs stream into the Input Sensory Vectors Shaping Modules (Figure 2). One shaping module exists for each sensory system, e.g., vision, auditory, and so on. The shaping module normalizes the sensory inputs into a form which can be used within the architecture. The normalized sensory inputs then move to the Input Sensory Vectors Association Modules. Here sensory features are spatially mapped onto navigation maps dedicated to one sensory system. Again, there is one association module for each sensory system, e.g., vision, auditory, and so on. The navigation maps holding sensory inputs of a given sensory system (e.g., vision) are matched against the best matching stored navigation map in that (e.g., visual) Input Sensory Vectors Association Module. That best matching stored navigation map is retrieved and then updated with the actual sensory input of that sensory system (or a new navigation map is created if there are too many changes). As such, there is a type of predictive coding occurring—the architecture anticipates what it is sensing and then considers the differences with the actual input signal. This works well for the perception of noisy, imperfect sensory inputs.

The navigation maps are then propagated to the Object Gateway Modules (Figure 2) where portions and the entire navigation maps are matched against stored multisensory navigation maps (i.e., contain features from multiple sensory systems) in the Causal Memory Module. The best matched navigation map from the Causal Memory Module is then considered moved to the Navigation Module. This best matching navigation map is updated with the actual sensory information sensed from the environment. The updated navigation map is termed the Working Navigation Map. An example of a Working Navigation Map is shown in Figure 3.

Instinctive primitives (pre-programmed) and learned primitives (learned), are small procedures that can perform operations on the Working Navigation Map. They are selected by a similar matching process in terms of what are the sensory inputs as well as in terms of signals from the Goal/Emotion Module and from the previous values of the Navigation Modules. The arrow (i) in Figure 2 shows the actions of the best matching instinctive primitive on the Working Navigation Map in the Navigation Module. These operations are essentially matrix operations such as comparing arrays, adding a vector to an array and other straightforward operations that could be expected of brain-inspired circuitry. The result is an output signal to the Output Vector Association Module and then to the Output Vector Shaping Module. This results in the activation of an actuator or the transmission of an electronics

communication signal. Then the cognitive cycle repeats—sensory inputs are processed again, the Navigation Module may produce an output action, and the cognitive cycle repeats again, and so on.

Feedback pathways, only partially shown in Figures 1 and 2, exist throughout the architecture. As noted above, there is a type of predictive coding occurring—the architecture anticipates what it is sensing and then considers the differences with the actual input or intermediate signal. This is advantageous for the perception of noisy, imperfect sensory inputs. In the prior literature it has been shown how by enhancing some of these feedback pathways, causal reasoning, analogical inductive reasoning, and compositional language readily emerge from the architecture (Schneider, 2023, 2024). The mechanisms behind these emergent properties are discussed in more detail below.

In this paper, it is asked what if the evolution of the human brain were to continue as it has in the past, and given an environment for such evolution, what advantageous changes could occur as reflected in a model such as the Causal Cognitive Architecture? More specifically, the paper considers an evolution where there are increased intelligence abilities (e.g., Legg and Hutter, 2007; Wang, 2019; Adams et al., 2012; or the ability to better solve complex problems which humans encounter in their lives).

2 Previous Work: The Causal Cognitive Architecture

A vast number of cognitive architectures exist (Samsonovich, 2010; Kotseruba, Tsotsos, 2020). However, given that this paper considers the further biological evolution of the brain modeled via the Causal Cognitive Architecture, this section focuses on the Causal Cognitive Architecture. The purpose of this section is to review the architecture so that the reader is provided with an understanding of its properties and operation, prior to the new work discussed in this paper in the following section. In the last sub-section of this section, the Causal Cognitive Architecture is then compared with other existing cognitive architectures as well as previous work on cognitive maps.

2.1 Sensory Inputs

In this section the Causal Cognitive Architecture is considered in more detail. Appendix A (Supplementary Material section) gives a more formal description of the Causal Cognitive Architecture. Modified equations are used to describe the architecture. They are “modified” in the sense that many of the equations contain pseudocode. A pseudocode is a common language (e.g., English) description of the logic of a software routine (Kwon, 2014; Olsen, 2005). Traditional pseudocode tends to reproduce program structure although in English, i.e., constructs such as While, Repeat-Until, For, If-Then, Case, and so on. However, doing so to describe the Causal Cognitive Architecture would be quite lengthy. The use of the more abstracted pseudocode in the equations describing the architecture provides a more understandable description of the architecture without sacrificing much accuracy.

The subject of this paper is the CCA7 version of the architecture. However, the operation of the CCA5 (Causal Cognitive Architecture 5) version of the architecture will first be described. Then in the sections further below, the evolution to the CCA6 (Casual Cognitive Architecture 6) version will be briefly considered. Then the focus of the paper will be on the evolution, operation, and properties of the CCA7 version of the architecture. The early sections of the formal description of the Causal Cognitive Architecture 7 (CCA7) in Appendix A (Supplementary Material section) apply to the CCA5, CCA6 and CCA7 versions of the architecture.

An overview of the CCA5 version of the architecture is shown in Figures 1 and 2. It is seen that sensory inputs stream into the Input Sensory Vectors Shaping Modules, with one module for each sensory system, e.g., vision, auditory, etc. As noted above, the Causal Cognitive Architecture works in terms of cognitive cycles. These cycles are biologically inspired. For example, Madl and colleagues (2011) hypothesize the essence of human cognition to be cascading cycles of operations. In the CCA5, each cognitive cycle, whatever information in the previous time period of the previous cognitive cycle has streamed into the Input Sensory Vectors Shaping Modules (arrow “a”, Figure 2) is further processed, normalized and then propagated to the Input Sensory Vectors Association Modules (arrow “b”, Figure 2), as well as to the Sequential/Error Correcting Module (arrow “c”, Figure 2).

The details of sensory perception, i.e., sensory signal processing, from the quantum level to the output produced by a transducer after possibly multi-layered initial signal processing, are largely abstracted away in this formalization. This does not diminish the importance of better signal processing. However, the architecture is concerned with whatever processed sensory inputs stream in, and that is what is considered here.

Equations (9) and (10) from Appendix A, are shown below. $\mathbf{S}_{1,t}$ is an array of sensory inputs of sensory system 1 (visual in the current simulation) which has accumulated since the last cognitive cycle to the present cognitive cycle at time t . $\mathbf{S}_{2,t}$, $\mathbf{S}_{3,t}$, and so on, are arrays of sensory inputs from other sensory systems. `Input_Sens_Shaping_Mods.normalize` is a method operating on arrays \mathbf{S} in the Input Sensory Shaping Modules, processing and normalizing the raw input sensory data. \mathbf{S}' is a normalized array, i.e., the raw array of the streams of sensory inputs for that sensory system has been normalized to a size and form that allows straightforward operations with the navigation maps within the architecture which are also arrays. The vector $\mathbf{s}'(t)$ holds the normalized arrays of sensory inputs at time t for the different sensory systems.

$$\mathbf{s}(t) = [\mathbf{S}_{1,t}, \mathbf{S}_{2,t}, \mathbf{S}_{3,t}, \dots, \mathbf{S}_{\theta-\sigma,t}] \quad (9)$$

$$\mathbf{s}'(t) = \text{Input_Sens_Shaping_Mods.normalize}(\mathbf{s}(t)) = [\mathbf{S}'_{1,t}, \mathbf{S}'_{2,t}, \mathbf{S}'_{3,t}, \dots, \mathbf{S}'_{\theta-\sigma,t}] \quad (10)$$

In simulations of the CCA5, CCA6 and CCA7 versions of the architecture, visual, auditory, and olfactory simulated inputs have been considered. However, the architecture is very flexible in accepting any number of different sensory system inputs. For example, a radar sensory system (or most other artificial sensory systems) could easily be added to the architecture – its inputs once processed and normalized will be treated as any other sensory inputs. Similarly, a particular sensory system can be rendered inoperable, and if the other sensory systems are sufficient, often there will be limited impact on the architecture’s final perception and behavior. As will be seen below, there is a very flexible approach towards the processing of sensory input data from a number of different sensory systems.

The Causal Cognitive Architecture is inspired by the mammalian brain. However, in order to simplify the system of equations used to model the architecture, the olfactory sensory system and any additional non-biological senses are treated the same as the other senses which in the brain relay through the thalamus to the neocortex. Similarly, the architecture does not model the left-right sides and the interhemispheric movement of working memory which occur in the biological brain.

As can be seen from Figure 1, outputs from the Input Sensory Vectors Shaping Module are also propagated to the Autonomic Reflex Modules. These Reflex Modules perform straightforward

actions in response to certain input stimuli, analogous to reflex responses in mammals. In this paper, there is more of a focus on the higher-level cognition occurring in the architecture. Similarly, this paper does not fully consider or model the repertoire of lower-level learning and behavior routines that exist in humans.

Vector $s'(t)$ (equation 10), holding the normalized arrays of sensory inputs is propagated to the Input Vectors Association Modules (arrow “b”, Figure 2) and to the Sequential/Error Correcting Module (arrow “c”, Figure 2). Spatial binding of the sensory inputs will occur in the Input Vectors Association Modules, i.e., each sensory system’s inputs will be mapped onto a navigation map (essentially an array) where the spatial relationship of different sensory inputs are maintained to some extent (e.g., the navigation map in Figure 3 which represents the sensory scene of Figure 6A).

In the Sequential/Error Correcting Module temporal binding of the sensory inputs occurs—temporal relationships of different sensory inputs, i.e., those of the current cognitive cycle time $t=t$ and those of previous cycles $t=t-1$ (i.e., 1 cognitive cycle ago), $t=t-2$, $t=t-3$ are bound spatially onto a navigation map and later combined with the spatial navigation maps (Schneider, 2022a, 2022b). This is discussed in more detail in Section A-4 in Appendix A (Supplementary Material). The need to store snapshots of the input sensory navigation maps every thirtieth of a second, for example, in order to track and make memories of motion (as well as changes in higher level cognitive processes) is obviated by transforming the temporal changes into a spatial features that are then mapped on to the same navigation map with the other spatial features.

Temporal binding is an essential feature of the Causal Cognitive Architecture. As mentioned above, it is described in more detail in Appendix A (in the Supplementary Material section) as well as the literature (Schneider, 2022a). If there is motion of an object, temporal binding allows the creation of a vector representing this motion which then can be mapped onto the existing spatial binding navigation map representing the input sensory data. As well, motion of ideas, i.e., changes in navigation maps, can also be similarly mapped onto other navigation maps. However, this paper will not focus on temporal binding, but rather consider in more detail the flow of sensory inputs $s'(t)$ into the Input Sensory Vectors Association Modules and then to Navigation Module and related modules (Figure 1).

Equation (18) from Appendix A is shown below. In each Input Sensory Vectors Association Module (there is a separate one for each sensory system) the sensory input features ($S'_{\sigma,t}$) for that sensory system σ are spatially mapped onto a navigation map. Each such navigation map is then matched (“match_best_local_navmap”) against the best navigation maps stored in that Input Sensory Vectors Association Module for that sensory system σ (*all_maps* $_{\sigma,t}$). That best matching stored navigation map is retrieved and then updated with the actual sensory input of that sensory system (or a new navigation map is created if there are too many changes). $\mathbf{LNM}_{(\sigma, \gamma, t)}$ is the updated navigation map (where σ is the sensory system, γ is the map address, and t is the time corresponding to the current cognitive cycle). \mathbf{LNM} stands for “Local Navigation Map” referring to this being the updated input sensory navigation map for that local sensory system (e.g., vision, auditory, olfactory, and so on). The vector \mathbf{lnm}_t (23) contains the best matching and updated navigation maps for each sensory system.

In predictive coding, the brain or artificial agent makes a prediction about the environment and then this prediction is sent down to lower levels of sensory inputs. Actual sensory inputs are compared to the prediction and the prediction errors are then used to update and refine a future prediction.

Essentially the brain or the artificial agent effectively functions to minimize prediction errors (Rao, Ballard, 1999; Friston, 2010; Millidge, Seth, Buckley, 2022; Georgeon, Lurie, Robertson, 2023).

In equation (18) \mathbf{WNM}'_{t-1} refers to the Working Navigation Map, i.e., the navigation map the architecture was operating on in the Navigation Module (Figure 1) in the previous cognitive cycle $t=t-1$, which will influence the matching of sensory inputs to the best local sensory navigation map stored in the particular Input Sensory Vectors Association module in the current cognitive cycle $t=t$. The architecture anticipates to some extent what it will be sensing and then considers the differences with the actual input signal, i.e., which stored navigation map of that sensory system's stored navigation maps (*all_maps* _{σ,t}) is the closest match based on the actual sensory input ($\mathbf{S}'_{\sigma,t}$) and based on what the Navigation Module expects to see (\mathbf{WNM}'_{t-1}). The architecture essentially matches sensory inputs with navigation maps it already has experience with, and then considers the differences (i.e., the error signal), with updated navigation maps (i.e., error signal resolved) saved in memory. This works well for the perception of noisy, imperfect sensory inputs, and this is easy to implement with the navigation map data structure (e.g., Figure 3) used by the architecture.

While this process has a number of similar aspects with predictive coding, it was not designed as a predictive coding architecture. For example, the architecture does not actively attempt, for example, to minimize free energy or minimize prediction errors, although this effect often results. Rather, this arrangement emerged in the attempt to model the evolution of the brain, albeit in a loosely functionalist approach (e.g., Lieto, 2021b).

$$\mathbf{LNM}_{(\sigma, \gamma, t)} = \text{Input_Assocn_Mod}_{\sigma}.\text{match_best_local_navmap}(\mathbf{S}'_{\sigma,t}, \text{all_maps}_{\sigma,t}, \mathbf{WNM}'_{t-1}) \quad (18)$$

$$\mathbf{lnm}_t = [\mathbf{LNM}_{(1, \gamma, t)}, \mathbf{LNM}_{(2, \gamma, t)}, \mathbf{LNM}_{(3, \gamma, t)}, \dots, \mathbf{LNM}_{(\theta-\sigma, \gamma, t)}] \quad (23)$$

The Local Navigation Maps (i.e., the best matching navigation maps for each “local” sensory system updated by the actual sensory inputs, represented by \mathbf{lnm}_t in (23)) are then propagated to the Object Gateway Modules (arrow “d”, Figure 2), with one module for each sensory system. In the Object Gateway Modules, portions of each navigation map will be attempted to be segmented into different objects, as best it can do. If there is, for example, a black sphere in a sensory scene, the Object Gateway Module when processing the visual LNM (local navigation map) will readily segment out a black sphere from the rest of the sensory scene. (The navigation map shown in Figure 3 was constructed in this manner from various visual closeup sensory inputs. For example, in cell (0,0,0) the link shown, i.e., {0024, 0,0,0} refers to another navigation map where the various lines and colors of a sphere were extracted from the navigation map as a black sphere, and where the descriptive labels “sphere” and “black” were linked to.)

The temporal binding (i.e., motion) of the sensory inputs that has occurred in the Sequential/Error Correcting Module is spatially mapped to each of the Local Navigation Maps (arrow “e”, Figure 2). For example, equation (62) taken from Appendix A (Supplementary Material), shows the Local Navigation Map for the visual sensory inputs $\mathbf{LNM}_{(1, \gamma, t)}$ being updated with a “Vector Navigation Map” \mathbf{VNM}'_t (i.e., the motion prediction vector created in the Sequential/Error Correcting Module and applied to an array) with the updated Local Navigation Map $\mathbf{LNM}'_{(1, \gamma, t)}$ resulting.

$$\mathbf{LNM}'_{(1, \gamma, t)} = \mathbf{LNM}_{(1, \gamma, t)} \cup \mathbf{VNM}''_t \quad (62)$$

The different sensory system updated Local Navigation Maps $\mathbf{LNM}'_{(1, \dots, \gamma, t)}$ are then matched against the multisensory (i.e., have features from all sensory systems as well as perhaps other features created and stored on the maps) navigation maps stored in the Causal Memory Module (arrow “f”, Figure 2). \mathbf{WNM}_t is the best matching multisensory navigation map. Equation (67) from Appendix A shows that the Object Segmentation Gateway Module (“Object_Seg_Mod”) built-in method (i.e., part of the circuitry of the Object Segmentation Gateway Module) “differences” compares the number of differences between actual sensory information on the Local Navigation Maps represented by *actual*_t to the features represented by \mathbf{WNM}_t .

As (67) shows, if the number of differences is low enough, then the best matching multisensory navigation map \mathbf{WNM}_t is updated with actual sensory information from the Local Navigation Maps represented by *actual*_t. (If there are too many differences between the best matching map and the actual input maps, i.e., greater than h' as shown in (67), then in another equation in Appendix A there will be the creation of a new multisensory navigation map and updating it, rather than the modification of the existing \mathbf{WNM}_t .) The resulting multisensory navigation map \mathbf{WNM}'_t is called the “Working Navigation Map” and is the navigation map upon which instinctive primitives and learned primitives operate on in the Navigation Module (arrow “g”, Figure 2). Figure 3 is an example of a Working Navigation Map.

$$\begin{aligned} & | \text{Object_Seg_Mod.differences}(\mathbf{actual}_t, \mathbf{WNM}_t) | \leq h', \\ & \Rightarrow \mathbf{WNM}'_t = \mathbf{WNM}_t \cup \mathbf{actual}_t \quad (67) \end{aligned}$$

2.2 The Navigation Module(s)

In the CCA5 version of the architecture there is a single Navigation Module (Figure 1). However, the Navigation Modules are increased in the CCA6 version (Figure 5) and the CCA7 version (Figure 9). Nonetheless, this section applies to all these versions of the architecture.

Instinctive primitives are small procedures that can perform operations on the Working Navigation Map (\mathbf{WNM}'_t) in the Navigation Module. Instinctive primitives are pre-existing—they come preprogrammed with the architecture. Learned primitives similarly are small procedures that can perform operations on the Working Navigation Map (\mathbf{WNM}'_t). However, learned primitives are learned by the architecture, rather than being preprogrammed.

The instinctive primitives are inspired by the instinctive behaviors present in human infants and children, as well as in some nonhuman primates (Spelke, 1994; Kinzler, Spelke, 2007). Spelke groups these instinctive behaviors in terms of physics of objects, agents, numbers, geometry, and reasoning about social group members.

As can be seen from Figure 2 (arrow “h”), the processed Input Sensory Vectors Association Modules’ navigation maps, as well as inputs from other parts of the architecture, propagate to the store of both learned primitives and instinctive primitives in the architecture. Either an instinctive primitive or a learned primitive will be selected. The primitive is selected by a similar matching process to the one discussed above, but here in terms of what are the sensory inputs as well as in

terms of signals from the Goal/Emotion Module and from the previous values of the Navigation Modules.

Arrow “i” in Figure 2 shows the best matching instinctive primitive acting on the Working Navigation Map (WNM_i) in the Navigation Module. These operations are essentially matrix/tensor operations such as comparing arrays, adding a vector to an array and other straightforward operations that could be expected of brain-inspired circuitry. Equation (82) is taken from Appendix A. In the Navigation Module the best matching primitive (it can be either an instinctive primitive or a learned

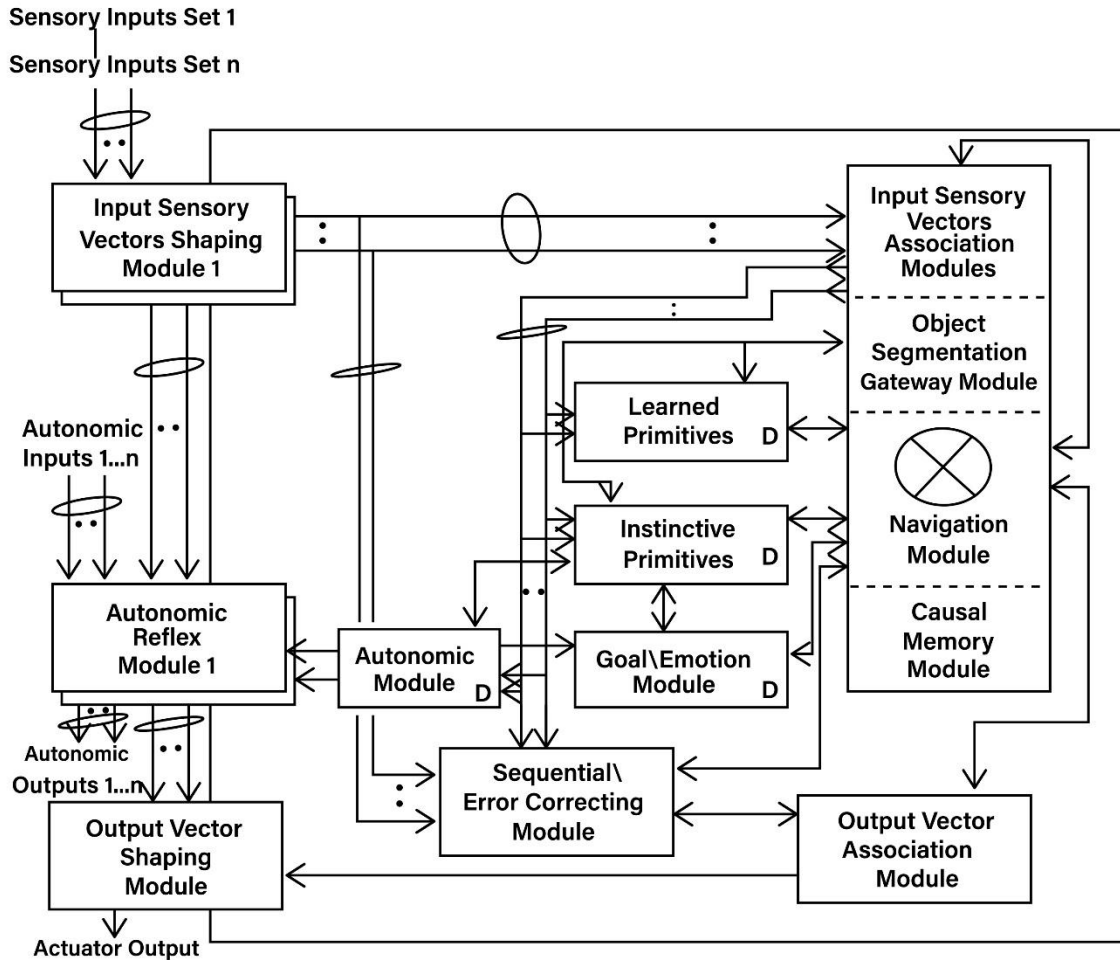


Figure 1. Overview of the Causal Cognitive Architecture 5 (CCA5) See text for description of modules and their operation.

(“D” indicates module has developmental function, i.e., changes algorithms with experience; ovals indicate pathways that are $n=0,1,2,\dots$ where there are sets of modules providing signals.)

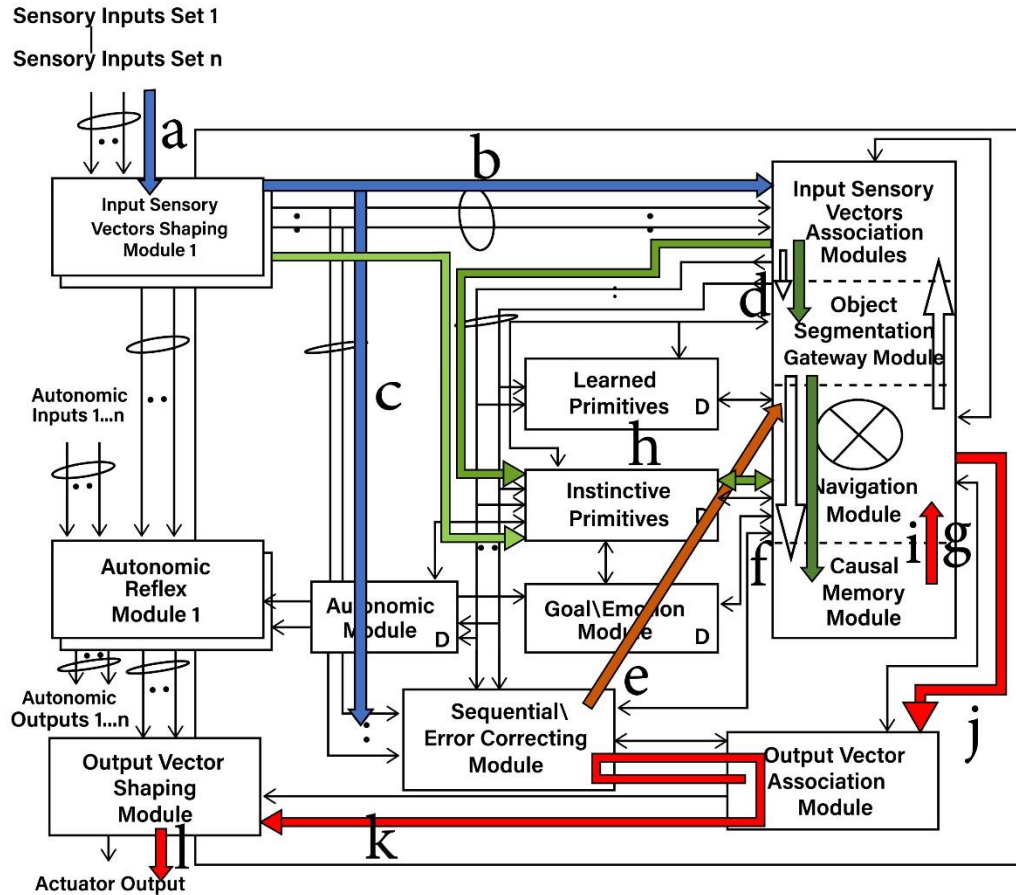


Figure 2. Cognitive Cycle: Sensory Inputs are Processed and an Output Occurs

a – Sensory inputs stream into the Input Sensory Vectors Shaping Modules

b – Processed and normalized sensory inputs propagate to the Input Sensory Vectors Association Modules and best-matching Local Navigation Maps (LMN's) for each sensory system produced (spatial binding)

c – Processed and normalized sensory inputs propagate to the Sequential/Error Correcting Module (temporal binding)

d – Segmentation of objects in the input sensory scenes

e – Spatial mapping of the temporal mapping of the sensory inputs from the Sequential/Error Correcting Module

f – Match to the best-matching Multi-Sensory Navigation Map from the Causal Memory Module producing the Working Navigation Map (WNM) for the Navigation Module

g – Working Navigation Map (WNM) in/accessible by Navigation Module

h – Selecting a best-matching primitive (an Instinctive Primitive in this case)

i – Instinctive Primitive operating on the Working Navigation Map (WNM)

j – action signal produced by operation of Instinctive Primitive on WNM

k – **output_vector** signal, motion corrected by the Sequential/Error Correcting Module

l – Signal to actuators or for electronic transmission

air	air	air	air	air	air
air	air	air	air	air	air
air	air	air	air	air	air
air	air	air	air	air	air
air	air	cylinder, white, link {0023,0,0,0}	air	air	air
sphere, black, link {0024,0,0,0}	air	block, black, link {0022,3,3,0}	air	black, block, link {0021,0,0,0}	air

Figure 3. Example of a Navigation Map—the 6x6x0 spatial dimensions are shown containing sensory features and links to other navigation maps. This represents the sensory scene of Figure 6A below. Although this navigation map only contains visual sensory features, other navigation maps can contain combinations of visual, auditory, olfactory, and so on, sensory features.

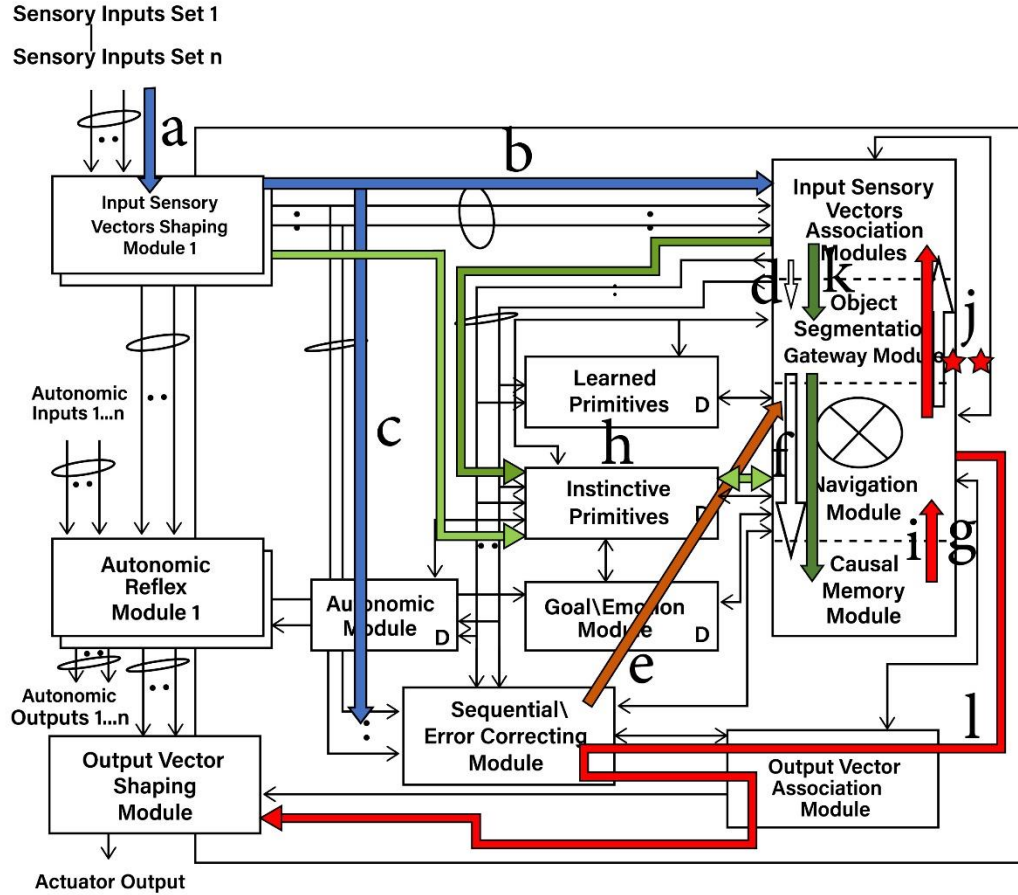


Figure 4. Consider the above figure under these situations:

A. (arrows $a \rightarrow j$) When the operation of the selected instinctive or learned primitive on the Working Navigation Map in the Navigation Module does not produce an output action, or a meaningful one, the results, i.e., the new contents of the Working Navigation Map, can instead be fed back to the Input Sensory Vectors Association Modules.

B. (arrows $k \rightarrow l$) Instead of actual sensory inputs the intermediate results from the Navigation Module (previous figure) which have been temporarily stored in the Input Sensory Vectors Association Modules, are now automatically propagated to the Navigation Module. In this new cognitive cycle perhaps a new instinctive primitive will end up being selected (or the same ones used) and applied to previous intermediate results, possibly producing a valid output action. If so, then output action goes to the Output Vector Association Module and then to the real world. However, if no valid output action occurs, the new intermediate results can be fed back again, and in the next cognitive cycle processed again.

C. (j^{})** (Note: Although not shown, assume there is a TempMap memory storage area in the Navigation Module, which is shown more explicitly in Figure 5.) No meaningful output was produced by the interaction of the primitive on the Working Navigation Map (WNM) in the Navigation Module. As before, the Working Navigation Map (i.e., the intermediate results) are fed back and stored in the Sensory Association Modules. However, the operations at j^{**} occur now (described below and in the text now occur). Effectively induction by analogy automatically occurs in these steps allowing the production of an actionable output in many situations.

a – Sensory inputs stream into the Input Sensory Vectors Shaping Modules

b – Processed and normalized sensory inputs propagate to the Input Sensory Vectors Association Modules and best-matching Local Navigation Maps (LMN's) for each sensory system produced (spatial binding)

c – Processed and normalized sensory inputs propagate to the Sequential/Error Correcting Module (temporal binding)

d – Segmentation of objects in the input sensory scenes

e – Spatial mapping of the temporal mapping of the sensory inputs from the Sequential/Error Correcting Module

f – Match to the best-matching Multi-Sensory Navigation Map from the Causal Memory Module producing the Working Navigation Map (WNM) for the Navigation Module

g – Working Navigation Map (WNM) in/accessible by Navigation Module

h – Selecting a best-matching primitive (an Instinctive Primitive in this case)

i – Instinctive Primitive operating on the Working Navigation Map (WNM)

j – No meaningful *action* signal produced by operation of Instinctive Primitive on WNM, thus contents of WNM are fed back to the Input Sensory Vectors Associations Modules

k – In the new cognitive cycle the stored WNM, i.e., the previous cycle's intermediate results, are now reprocessed through the Navigation Module

l – Perhaps a meaningful *action* signal is now produced, and an output signal results

j** –No meaningful *action* signal produced by operation of Instinctive Primitive on WNM, thus the following happens (represented above by ** since there is not enough space to show the various arrows required):

1. the contents of WNM are fed back to the Input Sensory Vectors Associations Modules, like before
2. the contents of WNM are also fed to triggering a match with the best matching nav map in the Causal Memory Module (**WNM'**_{*best_match*})
3. the nav map that **WNM'**_{*best_match*} links to is placed in “**TempMap**” memory of the Navigation Module
4. the difference (**WNM'**_{*difference*}) between **WNM'**_{*best_match*} and “**TempMap**” is kept in the Navigation Module
5. the original **WNM** being stored in the Input Sensory Vectors Association Modules in the next cognitive cycle propagates forward and is added to **WNM'**_{*difference*} resulting in a new Working Navigation Map **WNM** which will be processed again (i.e., action by primitive against it) in the new cognitive cycle, and this time may (or may not) result in an actionable output from the Navigation Map

primitive) **WPR**_{*t*}, is applied against the Working Navigation Map **WNM'**. (In the CCA6 and CCA7 versions of the architecture where there are more than one Navigation Module, this occurs in Navigation Module A.)

In (82) “**apply_primitive**” is a built-in method (i.e., part of the circuitry of the Navigation Module) which does the actual operations of applying the primitive **WPR**_{*t*} on the Working Navigation Map **WNM'**. The result is an output signal *action*_{*t*} (82). As indicated by arrow “j” in Figure 2 output signal *action*_{*t*} is propagated to the Output Vector Association Module. As indicated by arrow “k” in Figure 2, there is modification of the output signal *action*_{*t*} with respect to temporal motions introduced in the Output Vector Association Module creating an intermediate **output_vector**_{*t*} signal. This is further modified by the **motion_correction**_{*t*} signal produced by the Sequential/Error Correcting Module. The result is the **output_vector**_{*t*} signal as shown in (87). The **output_vector**_{*t*} signal propagates to the Output Vector Shaping Module. The signal is further transformed for interface with the real world (arrow “l”, Figure 2), where it can result in the activation of an actuator or the transmission of an electronics communication signal.

$$action_t = \text{Nav_ModA.apply_primitive}(\mathbf{WPR}_t, \mathbf{WNM}'_t) \quad (82)$$

$$\begin{aligned} \mathbf{output_vector}'_t &= \text{Output_Vector_Mod.} \\ &\text{apply_motion_correction}(\mathbf{output_vector}_t, \mathbf{motion_correction}_t) \end{aligned} \quad (87)$$

Then the cognitive cycle repeats—sensory inputs are processed, the Navigation Module may produce an output action which results in an actuator output, and then the cycle repeats again, and so on.

Step by step examples of the architecture processing a particular sensory scene are given in Schneider (2022a, 2023). As well, more details in the processes occurring are given in Appendix A (Supplementary Material section).

2.3 Feedback Operations

Feedback pathways, only partially shown in Figures 1 and 2, exist throughout the Causal Cognitive Architecture. These feedback pathways are essential—the architecture considers the differences with the actual input or intermediate signal compared to its existing internal representations, as discussed above. This is advantageous for the perception of noisy, imperfect sensory inputs.

Schneider (2022a) describes how by enhancing some of these feedback pathways, causal properties readily emerge from the architecture. In Figure 4A the feedback pathways from the Navigation Module back to the Input Sensory Vectors Association Modules are enhanced. Biologically, such a change could have occurred in the evolution of the brain through a number of genetic mechanisms (e.g., Rakic, 2009; Chakraborty and Jarvis, 2015).

The result of this enhancement of this feedback pathway is that when the operation of the instinctive or learned primitive on the Working Navigation Map in the Navigation Module does not produce an output action, or a meaningful output action, the results, i.e., the new contents of the Working Navigation Map, can instead be fed back to the Input Sensory Vectors Association Modules instead of being sent to the Output Vector Association Module.

As shown in Figure 4A, the Navigation Module in this cognitive cycle did not produce any output signal $action_t$. However, as arrow “j” (Figure 4A) shows, the contents of the Navigation Module are fed back to the Input Vectors Association Module. These saved contents essentially now represent intermediate results of the Navigation Module. In the next cognitive cycle they can be fed forward to the Navigation Module, and operated on again.

As Figure 4B shows, in the next cognitive cycle, instead of processing the actual sensory inputs, these stored intermediate results, will automatically propagate forward to the Navigation Module, and be processed again (arrow “k”). These saved, essentially intermediate results become again the current Working Navigation Map (**WNM'**) in the Navigation Module. The advantage of reprocessing intermediate results is that another operation of the instinctive (or learned) primitive on these results can often result in a better, actionable output signal $action_t$ (82). If not, the new intermediate results (i.e., the new contents of the Working Navigation Map) can be fed back and again be re-processed in the next one or many repeated cognitive cycles. (At present, determination of what is a meaningful result can be determined in part by a learned or instinctive primitive's procedures, as well as in part if the action signal sent to the Output Vector Association Module can be acted upon.)

Equations (88) and (89) taken from Appendix A show that if the action signal produced by the Navigation Module is not actionable (i.e., $action_t \neq \text{“move*”}$) then the Working Navigation Map in the Navigation Module is fed back to the various sensory system Input Vector Association Modules (88). In the next cognitive cycle $t=t+1$ the best-matching Local Navigation Map in each module now becomes the sensory features extracted from the fed back Working Navigation Map (89) and so, these Local Navigation Maps will be automatically propagated forward to the Navigation Module, where they will constitute the Working Navigation Map again. Thus, intermediate results of the

previous cycle will be operated on a second time by whatever instinctive or learned primitives are selected this cognitive cycle.

While this seems like an inelegant way to re-operate on intermediate results, to evolve this mechanism in the brain takes modest changes, i.e., enhancement of particular feedback pathways. Indeed, in humans when the brain switches from the automatic operations of System 1 to the more complex operations of System 2 (Kahneman, 2011), which is similar to re-operating on intermediate results, less attention can be paid to the normal stimuli around us, which is what happens in the Causal Cognitive Architecture during re-operating on intermediate results.

$$\begin{aligned}
 & (action_t \neq \text{"move*"} \text{ and } \mathbf{WPR}_t \neq [\text{"discard*"}]) \text{ or } \mathbf{WPR}_t = [\text{"feedback*"}], \\
 & \{ \Rightarrow \text{Nav_ModA.feedback_to_assocn_mod}(\mathbf{WNM}'_t) \} \quad (88) \\
 & \Rightarrow \forall_{\sigma} : \mathbf{LNM}_{(\sigma, \gamma, t+1)} = \\
 & \text{Input_Sens_Vectors_Assoc_Module}_{\sigma}.\text{extract}_{\sigma}(\mathbf{WNM}'_t) \} \quad (89)
 \end{aligned}$$

Schneider (2022a) shows that by re-operating on the intermediate results, the architecture can generate casual behavior by exploring possible cause and effects of the actions. An example is given where the CCA3 version of the Causal Cognitive Architecture is controlling a hospital patient assistant robot. A new robot has never seen a patient fall to the ground and has never been trained on a closely identical case. However, it has a learned primitive (i.e., procedure) from a rudimentary education prior to doing this work that it should not allow any patient it is with to fall down on the ground.

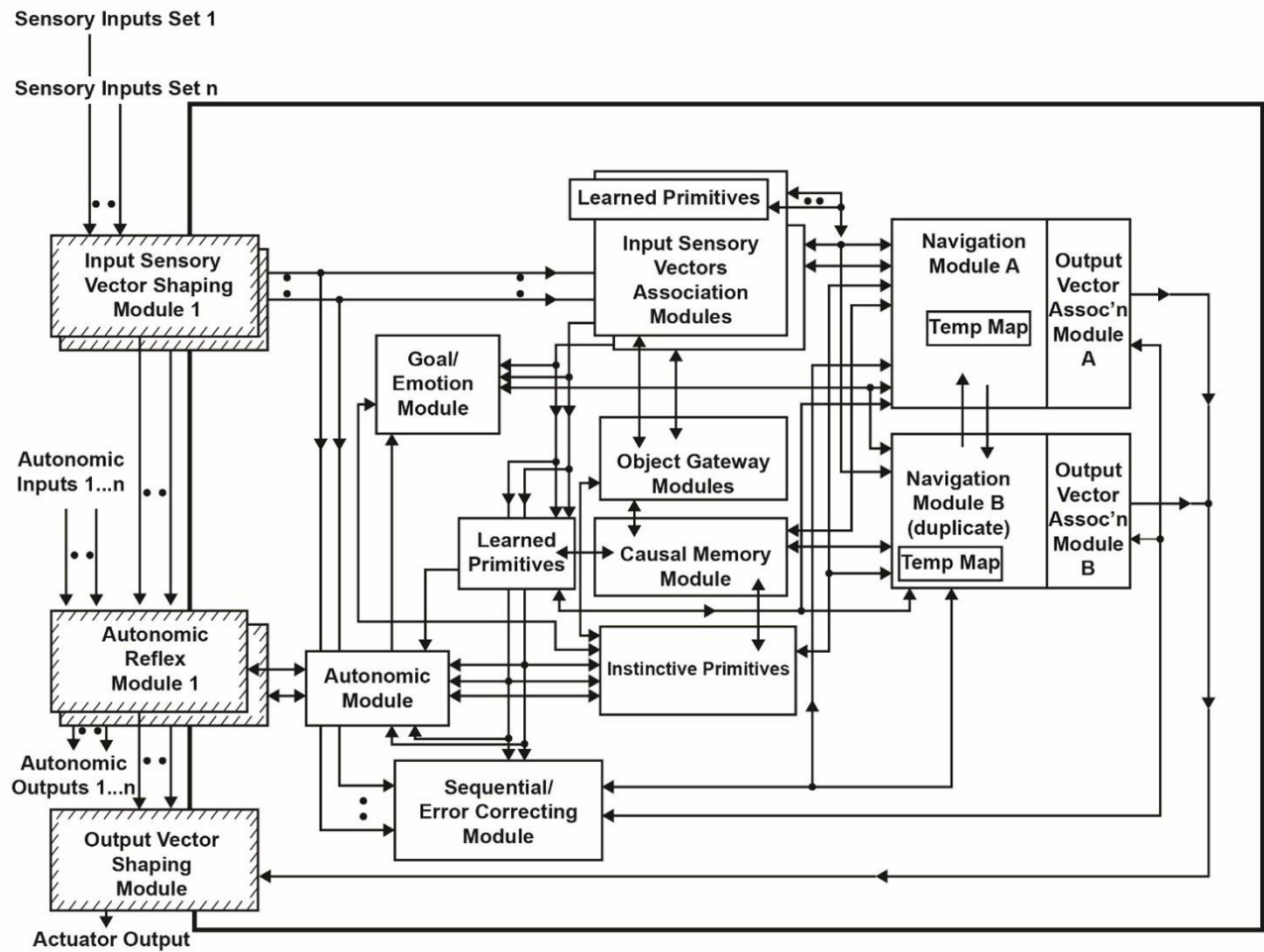
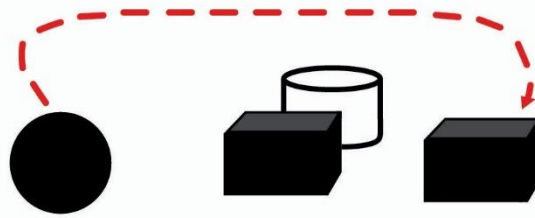


Figure 5. The Causal Cognitive Architecture 6 (CCA6). The Navigation Module of the CCA5 architecture has been duplicated into Navigation Module A and Navigation Module B, along with other changes, resulting in the CCA6 version of the architecture.

A



B



air	air	air	air	air	air
air	air	air	air	air	air
air	air	air	air	air	air
air	air	air	air	air	air
air	air	cylinder, white, link {0023,0,0,0}	air	air	air
sphere, black, link {0024,0,0,0}	air	block, black, link {0022,3,3,0}	air	black, block, link {0021,0,0,0}	air

C



"not", link {+}	"near", link {+}	"a", link {+}	"cylinder", link {+}		
"of", link {+}	"the", link {+}	"black", link {+}	"block", link {+}	"which", link {+}	"is", link {+}
"place", link {+}	"the", link {+}	"black", link {+}	"sphere", link {+}	"on", , link {+}	"top", link {+}

Figure 6. **A.** Connectionist systems have difficulty solving problems such as “place the black sphere on top of the black block which is not near a cylinder.” (*The arrow shows the correct solution to this problem. Of course, it is not shown to the system being asked to solve this problem.*)

B, C. The sensory scene of Figure 6A is loaded in Navigation Module A (Figure 6B). The instruction associated with Figure 6A (“place the black sphere on top of the black block which is not near a cylinder”) is loaded in Navigation Module B (Figure 6C). In subsequent cognitive cycles the contents of Navigation Module B are processed against the contents of Navigation Module A via operations of various instinctive primitives, discussed in the text.

air	air	air	air	air
air	air	air	air	air
air	air	air	air	air
air	air	air	air	air
air	air	cylinder, white, link {0023,0,0,0}	air	air
<"place">,sphere, black, link {0024,0,0,0}	air	block, black, link {0022,3,3,0}	air	black, block, link {0021,0,0,0}

A - Navigation Map in Navigation Module A

air	air	air	air	air
air	air	air	air	air
air	air	air	air	air
air	air	air	air	air
air	air	<"not">,cylinder, white, link {0023,0,0,0}	air	air
<"place">,sphere, black, link {0024,0,0,0}	air	<"not">, <"top">,block, black, link {0022,3,3,0}	air	<"top">,black, block, link {0021,0,0,0}

B - Navigation Map in Navigation Module A

Figure 7. **A.** The instinctive primitive "parse_sentence()" has entered the tag <"place"> in the cell containing the black sphere.

B. After a few more cognitive cycles the instinctive primitive "parse_sentence()" and then the instinctive primitive "physics_near_object()" have now written these tags in the various cells of the navigation map in Navigation Module A

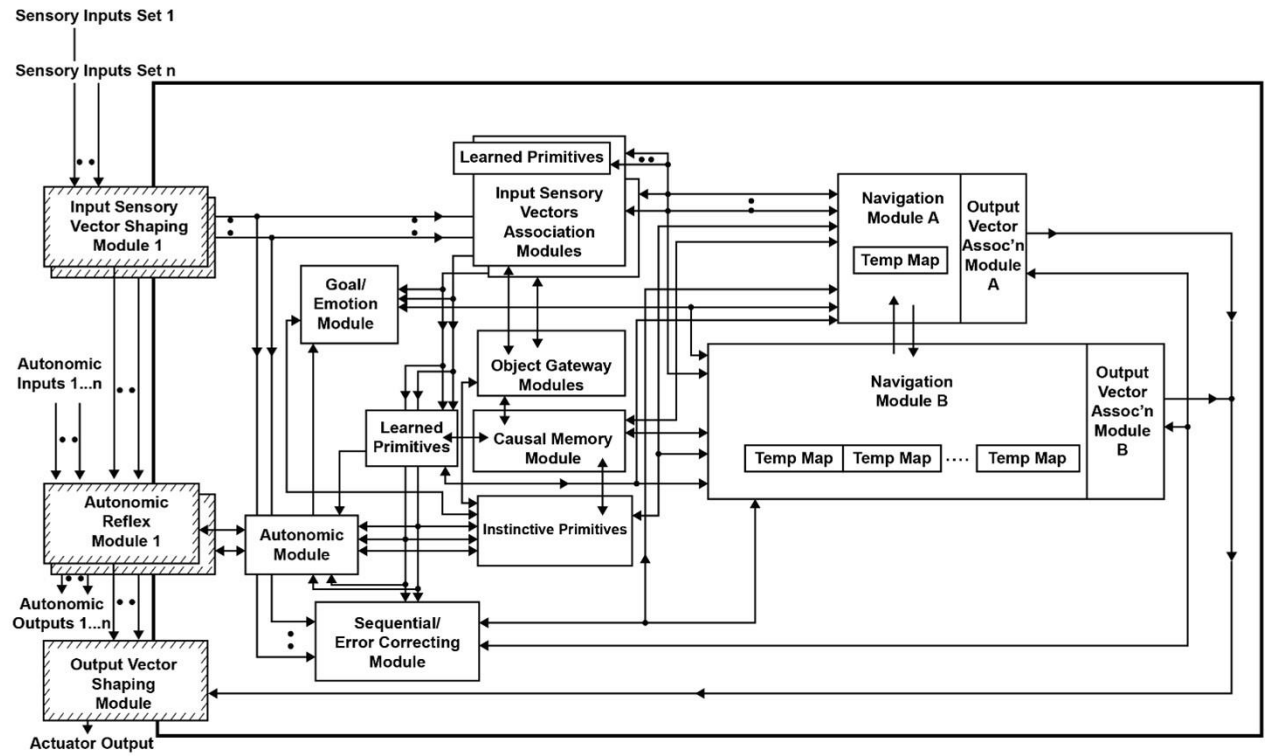


Figure 8. The CCA6 architecture with the duplication of the **TempMap** temporary memory areas in Navigation Module B.

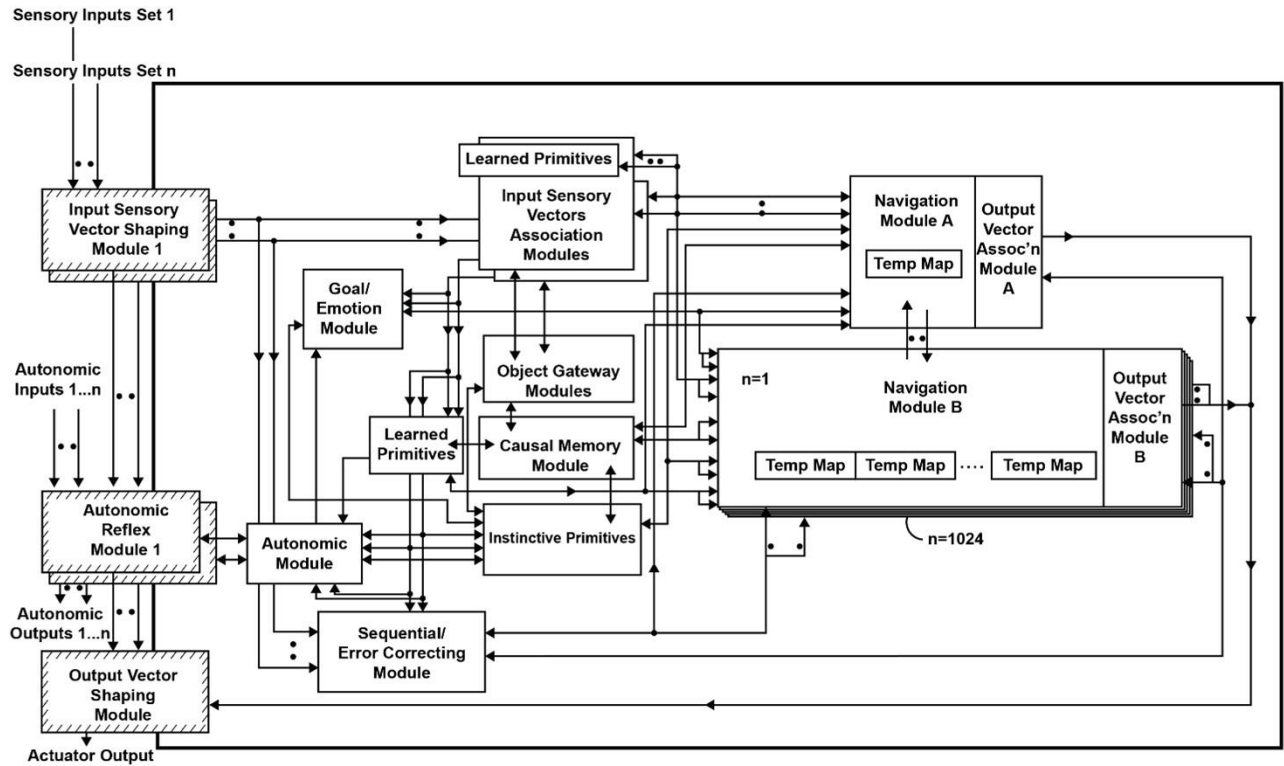


Figure 9. The Causal Cognitive Architecture 7 (CCA7). This is the architecture shown in Figure 8 now with duplication (1024 copies) of Navigation Module B.

The robot one day happens to be assisting a patient who begins to fall towards the ground. The learned primitive concerning a person falling is triggered but does not produce an actual output signal. The intermediate result calls the physics instinctive primitive (i.e., a general knowledge procedure pre-programmed in the architecture) which pushes back against something falling or moving in order to stop the movement. Thus, the robot pushes back against the falling patient, and stops the patient from falling, even though it has never actually done this before in training.

2.4 Analogical Reasoning

Even with reprocessing of intermediate results, there are still many situations where the Navigation Module will not produce any actionable output. Schneider (2023) shows that with biologically-feasible, modest modifications to the feedback operations, then analogical reasoning readily emerges. Although not explicitly shown, note that a temporary memory area, “**TempMap**” now also exists in the Navigation Module (Figure 4C). (This memory area is treated equivalently to an array in the equations, hence the bolding of its name.)

(Given the existence of a temporary memory area, why for causal behavior, as shown above, it is necessary to feed back the intermediate results of the Navigation Module to the Input Sensory Vectors Association Modules, rather than just store them in the temporary memory area? As Schneider (2023) notes, the reason is that the Causal Cognitive Architecture is biologically inspired, and from an evolutionary perspective, it seems more reasonable that storage of intermediate results

could occur by enhancing feedback pathways rather than by creating new memory areas. In order to efficiently carry out analogical operations, as described below, the evolutionary usurpation of some brain region as a temporary storage region would have been advantageous at this point. Thus, the CCA5 and higher versions of the architecture possess a temporary memory area associated with the Navigation Module.)

As before, the Working Navigation Map (\mathbf{WNM}'_t) (i.e., the intermediate results) are fed back and stored in the Input Sensory Association Modules (arrow “j” in Figures 4B, 4C). However, the Working Navigation Map is also propagated to the Causal Memory Module where it will automatically become matched with the best stored multisensory navigation map, which becomes the new Working Navigation Map in the Navigation Module. The navigation map that this navigation map recently linked to is triggered and retrieved, and then propagated to and subtracted from the new Working Navigation Map in the Navigation Module (“**” in Figure 4C). In the next cognitive cycle, as before, the original Working Navigation Map is automatically propagated, although now added to the differences in the Navigation Module. As a result of a few modifications to the feedback pathways and operations on the navigation maps here, effectively induction by analogy automatically occurs in these steps.

Equations (95) – (99) are taken from Appendix (Supplementary Material) and show these operations. If there is no actionable output from the Navigation Module (i.e., $action_t \neq \text{“move*”}$, where “move*” is an output signal giving instructions about moving something or moving a message), then these operations are automatically triggered, i.e., these equations occur. (Note that there is only one Working Navigation Map \mathbf{WNM}'_t in the Navigation Module (Figure 4C) at any time. However, since the contents of what is \mathbf{WNM}'_t change several times in these operations, for better clarity to the reader, a small descriptor is appended to its name, e.g., “ $\mathbf{WNM}'_{t-original}$ ”, etc.)

In (95) the contents of the Navigation Module, i.e., \mathbf{WNM}'_t which for better readability is called “ $\mathbf{WNM}'_{t-original}$ ” here, is fed back to be stored in the Input Sensory Vectors Association Modules, the same as before. However, “ $\mathbf{WNM}'_{t-original}$ ” is also propagated to the Causal Memory Module where automatically the “Causal_Mem_Mod.match_best_map” built-in method will occur matching it with the best matching multisensory map which becomes the new Working Navigation Map, which is called here “ $\mathbf{WNM}'_{t-best_match}$ ” (96).

In (97) the built-in method “Nav_ModA.use_linkaddress1_map” retrieves the navigation map that “ $\mathbf{WNM}'_{t-best_match}$ ” most recently linked to, and puts this map into the **TempMap** memory. Then as (98) shows the difference of “ $\mathbf{WNM}'_{t-best_match}$ ” and **TempMap** gets stored in the Navigation Module as “ $\mathbf{WNM}'_{t-difference}$ ”. Then a new cognitive cycle starts, and $\mathbf{WNM}'_{t-original}$ is automatically fed forward, but now added to “ $\mathbf{WNM}'_{t-difference}$ ” (99). The new Working Navigation Map \mathbf{WNM}'_t is termed “ $\mathbf{WNM}'_{t-analogical}$ ” since it represents an analogic inductive result.

Consider a navigation map \mathbf{x} which is represented by “ $\mathbf{WNM}'_{t-original}$ ”. Given that there was no actionable output in the last cognitive cycle, it is advantageous to induce what this navigation map \mathbf{x} will do next, i.e., which navigation map will it call. In (100) it is shown that it has properties/features $P_1 \dots P_n$. Consider navigation map \mathbf{y} represented as “ $\mathbf{WNM}'_{t-best_match}$ ” in (96). It is the best matching navigation map to navigation map \mathbf{x} and thus assumed it will share many properties, as shown in (101).

Navigation map **y** has previously linked to (i.e., it was pulled into the Navigation Module) the navigation map in **TempMap** (97), i.e., as given by the built-in method “use_linkaddress1_map”. (Schneider (2023) discusses other links and groups of links to use as the basis for analogical induction.) The difference between navigation map **y** and **TempMap** is “**WNM**_{*t*}-difference” (98). Thus, consider this difference, i.e., “**WNM**_{*t*}-difference” to be property N, as noted in (102).

Since navigation map **y** has property N, therefore by induction by analogy, it can be said that navigation map **x** also has property N (103). Thus, add property N, which is actually “**WNM**_{*t*}-difference”, to navigation map **x**, which is actually “**WNM**_{*t*}-original”, producing the result of navigation map **x** with property N as being “**WNM**_{*t*}-analogical” (99).

$$\begin{aligned}
 &((action_t \neq \text{“move”} \text{ or } \mathbf{WPR}_t = [\text{“analogical”}]) \text{ and } \mathbf{WPR}_t \neq [\text{“discard”}] \text{ and } \mathbf{WPR}_t \neq [\text{“feedback”}]), \\
 &\quad \{ \Rightarrow \text{Nav_ModA.feedback_to_assocn_mod}(\mathbf{WNM}'_{t\text{-original}}) \} \quad (95) \\
 &\Rightarrow \mathbf{WNM}'_{t\text{-best_match}} = \text{Causal_Mem_Mod.match_best_map}(\mathbf{WNM}'_{t\text{-original}}) \quad (96) \\
 &\Rightarrow \mathbf{TempMap}_t = \text{Nav_ModA.use_linkaddress1_map}(\mathbf{WNM}'_{t\text{-best_match}}) \quad (97) \\
 &\Rightarrow \mathbf{WNM}'_{t\text{-difference}} = \text{Nav_ModA.subtract}(\mathbf{WNM}'_{t\text{-best_match}}, \mathbf{TempMap}_t) \} \quad (98) \\
 &((action_{t-1} \neq \text{“move”} \text{ or } \mathbf{WPR}_{t-1} = [\text{“analogical”}]) \text{ and } \mathbf{WPR}_{t-1} \neq [\text{“discard”}] \text{ and } \mathbf{WPR}_{t-1} \neq [\text{“feedback”}]), \\
 &\quad \Rightarrow \mathbf{WNM}'_{t\text{-analogical}} = \text{Nav_ModA.retrieve_and_add_vector_assocn}() \quad (99) \\
 &\quad P_1\mathbf{x} \ \& \ P_2\mathbf{x} \ \& \ \dots \ P_n\mathbf{x} \quad (100) \\
 &\quad P_1\mathbf{y} \ \& \ P_2\mathbf{y} \ \& \ \dots \ P_n\mathbf{y} \quad (101) \\
 &\quad N\mathbf{y} \quad (102) \\
 &\quad \therefore N\mathbf{x} \quad \square \quad (103)
 \end{aligned}$$

As noted above, equations (100) – (103) essentially define induction by analogy. In (100) **x** has properties/features P_1 to P_n . **y** is similar and also has properties/features P_1 to P_n (101). **y** also has feature “N” (102). Thus, by induction by analogy, **x** has feature “N” (103). As shown above, a ready mechanism now exists in the Causal Cognitive Architecture which follows this definition. If an actionable resolution of a Working Navigation Map (**WNM**_{*t*}) does not immediately occur (i.e., a primitive applied to **WNM**_{*t*} does not produce an actionable output from the Navigation Module), the architecture can follow the analogical mechanism above to produce an analogical result which can be operated on in the next cognitive cycle.

Of interest, analogical intermediate results are useful in typical day-to-day functioning, rather than being considered as something only used in exceptional high-level problem-solving tasks (e.g., writing an intelligence test). For example, in Schneider (2023) there is an example of a robot controlled by a CCA5 version of the architecture. The robot needs to cross a river and has instinctive primitives that guide it to stay on solid ground to do so. However, there are piles of leaves floating on the river which appear solid and for which the robot has no experience about nor any instinctive primitives about. By analogical reasoning it is shown how the robot automatically uses a previous navigation map (i.e., experience) of stepping on pieces of newspaper floating in a puddle and its leg going into the puddle, to not use the leaves to cross the river. The robot has no knowledge

whatsoever about newspapers or leaves, other than they appear to be solid, yet by automatically using its analogical reasoning mechanism it successfully crosses the river via another path and not by stepping on the piles of floating leaves.

Hofstadter (2001) provides evidence that supports the use of analogies as the core of human cognition. Of interest, full analogical reasoning does not appear to be present in chimpanzees (Penn, Holyoak and Povinelli, 2008) although more recent reports show some animals capable of some aspects of analogical reasoning (Flemming et al., 2013; Hagmann et al., 2015). The mechanisms described above for the Causal Cognitive Architecture show theoretically that modest modifications can result in the emergence of analogical reasoning from a chimpanzee-human last common ancestor, albeit a loosely functionalist model thereof.

2.5 Compositionality

Given the usefulness of the Navigation Module, duplicating it into two Navigation Modules might be more advantageous. Again, biologically such a change could have readily occurred in the evolution of the brain through a number of genetic mechanisms (e.g., Rakic, 2009; Chakraborty and Jarvis, 2015). Figure 5 shows the duplication of the Navigation Module into Navigation Module A and Navigation Module B. This new version of the architecture is called the Causal Cognitive Architecture 6 (CCA6).

Consider the compositional problem shown in Figure 6A such as following the command to “place the black sphere on top of the black block which is not near a cylinder.” (The arrow shows the correct solution to this problem. Of course, the arrow is not shown to the system being asked to solve this problem.) Connectionist systems have trouble with such compositional problems. For example, Marcus et al. (2022) give a similar example to DALL-E2 and prompt it to place a red ball on top of a blue pyramid behind a car that is above a toaster. DALL-E2 tries ten times and produces various output images in response to the command, but none of these actually depict the requested relationships correctly.

However, Schneider (2024) shows that if the Navigation Module is duplicated into Navigation Module A and Navigation Module B, as shown in Figure 5, then compositionality and compositional language readily emerge from this CCA6 version of the architecture.

In the CCA6 architecture shown in Figure 5, compositional operations tend to occur in Navigation Module B. Instinctive primitives (as well as learned primitives) involved in compositional operations and language operations, will generally operate on the navigation map in Navigation Module B. Consider the example shown in Figure 6A of “placing the black sphere on top of the black block which is not near a cylinder.” The sensory scene of the spheres and blocks will propagate through the architecture (Figure 5) and be mapped to a navigation map in Navigation Module A, as shown in Figure 6B. (It actually takes a few cognitive cycles and close up views of the objects, as evidenced by the links in some of the cells (e.g., link to {map=24, x=0,y=0,z=0} for cell (0,0,0) with the labels “sphere”, “black”), to create this navigation map.)

Equations (109) – (114) are taken from Appendix A (Supplementary Materials). The instinctive primitive “`parse_sentence`” is triggered by the instruction (“*instruction sentence*”) associated with this sensory image. In (109) “`parse_sentence.copy()`” maps the instruction (“place the black sphere on top of the black block which is not near a cylinder”) to a navigation map (**WNMB’_t**) in Navigation Module B (“Nav_ModB”). This is shown in Figure 6B and Figure 6C. The “`link{+}`”

in the cells in the Navigation Map (**WNMB'**_{*t*}) in Navigation Module B just means that the cell links to its neighbor to the right.

The “`parse_sentence.parse()`” instinctive primitive parses through Navigation Map B **WNMB'**, i.e., the instruction sentence (110). Each word is matched against the Causal Memory Module “`parse_sentence.parse.match()`” (111). If what is called an “action word” is found (i.e., it matches some specific action to do to the other cells) then it is mapped to cells in Navigation Map A **WNM'** in Navigation Module A (Figure 7A) containing features associated with or mapping to the action word. In Figure 7A it can be seen, for example, that “place” has been matched to cell (0,0,0) in Navigation Module A. This makes sense since this action word is associated with the black sphere in the instruction sentence. (This is described in more detail in Appendix A.)

A “*near_trigger*” is a feature that is spatially near something else or not near something else that can trigger various physics instinctive primitives. The instruction sentence word “near” triggers instinctive primitive “`Nav_ModB.physics_near_object()`” (112). The result of this instinctive primitive is to place the tag “not” in the cells “not near” the white cylinder, as seen in the Navigation Map of Navigation Module A in Figure 7B.

Each cognitive cycle the CCA6 architecture will continue to parse through the instruction sentence. When it reaches the “*end_of_communication*” (i.e., the end of the sentence) it then parses through Navigation Module A looking for a “place” tag. If there is a “place” tag (e.g., cell (0,0,0)) in Navigation Module A in Figure 7B then instinctive primitive “`Nav_ModA.place_object()`” is triggered (113). This instinctive primitive will go through the navigation map looking for other tagged notations such as “top” in cells (2,0,0) and (4,0,0) in Navigation Module A (Figure 7B). It will ignore (2,0,0) since there is a “not” tag there but consider (4,0,0) valid. It will then trigger the instinctive primitive “`Nav_ModA.move()`” which then sends the action signals to the Output Vector Association Module A and which in turn sends a motion corrected signal to the Output Vector Shaping Module which instructs the actuators to move the black sphere to the cell (4,0,0) with the black block on the right (114).

```
( instruction_sentence ),
  {⇒WNMB't = Nav_ModB.parse_sentence.copy() (109)
    ⇒ Nav_ModB.parse_sentence.parse(WNMB't), (110)
      [⇒ Nav_ModB.parse_sentence.parse.match() (111)
        ⇒ near_trigger,
          (⇒Nav_ModB.physics_near_object()) ] (112)
    ⇒ end_of_communication,
      [ <place>,
        (⇒ Nav_ModA.place_object() (113)
          ⇒ Nav_ModA.move()) ] } (114)
```

Compositionality is a key property of an intelligent system. Without compositionality such a system would need to experience every (or very many) permutations of a vast number of sensory scenes and actions in order to learn them. Above it was shown how with the duplication of the navigation modules compositional abilities can readily emerge. This is discussed in more detail in Schneider (2024), including the emergence of language.

2.6 Comparison of the Causal Cognitive Architecture with other Cognitive Architectures

Samsonovich (2010) and Kotseruba and Tsotsos (2020) review the many cognitive architectures proposed in the literature. Kotseruba and Tsotsos note the large diversity of cognitive architectures proposed and the difficulty of defining the term. They consider cognitive architectures broadly as programs that “could reason about problems across different domains” as well as attempting to help determine what “particular mechanisms succeed in producing intelligent behavior” in terms of modeling the human mind.

Laird, Lebiere and Rosenbloom (2017) attempt to unify the field of cognitive architectures with what they term a “standard model of the mind.” In their standard model perception feeds into a working memory, while motor outputs feed out of it. There is bidirectional movement of information between a declarative long-term memory and the working memory. Similarly, there is bidirectional movement of information between a procedural long-term memory and the working memory. This is a very generic model of a cognitive architecture, and it would be expected to capture inclusion of most of the models listed by Samsonovich (2010) or Kotseruba and Tsotsos (2020). However, the CCA7 version of the Causal Cognitive Architecture surprisingly does not fit within this “standard model.”

In this standard model of the mind there are separate areas for declarative long-term memory and procedural long-term memory. However, in the CCA7, there can be both declarative long-term memory (i.e., navigation maps of experiences) and procedural long-term memory (i.e., instinctive and learned primitives) mixed together in the different Input Sensory Vectors Association Modules and within multisensory navigation maps which are operated on in the Navigation Modules A and B.

The CCA7 architecture is largely defined by its binding of sensory inputs into navigation maps and comparing these inputs with pre-stored information. The CCA7 architecture is also largely defined by its heavy usage of feedback of intermediate results of navigation maps. Again, these operations are not typical for most of the architectures defined by the standard model of the mind or included by Samsonovich (2010) or by Kotseruba and Tsotsos (2020).

2.7 Cognitive Maps

As noted above, the Causal Cognitive Architecture hypothesizes that the the navigation circuits in the amniotic ancestors of mammals duplicated many times to eventually form the neocortex. Thus, the millions of neocortical minicolumns are modeled in the Causal Cognitive Architecture as millions of navigation maps. As noted above, using this postulation it has been possible to show the emergence of causal reasoning, analogical reasoning and compositionality from a brain based on such navigation maps (Schneider, 2022a, 2023, 2024). (Albeit, not rigidly replicating the mammalian brain, but at a more functionalist system as per Lieto (2021b).)

Similar to the concept of navigation maps, is that of cognitive maps, which were proposed by Tolman (1948). A cognitive map is considered the way the brain represents the world and allows navigation and operations on paths and objects in the world. Thus, cognitive maps can hold geographical information as well as information from personal experiences. Prior to the work by O’Keefe and many others (e.g., O’Keefe, Nadel, 1978), there was much debate about the existence of cognitive maps in a large spectrum of the animal world. This debate still continues, for example, whether cognitive maps exist in insects (Dhein, 2023).

In mammals, experimental work has largely found evidence for cognitive maps existing in terms of spatial navigation (e.g., O’Keefe, Nadel, 1978; Alme et al., 2014). However, Behrens and colleagues (2018) and Whittington and colleagues (2022) review extensions of cognitive maps into other domains of cognition. Hawkins and colleagues (2019) note evidence within the mammalian neocortex for the equivalent of grid cells. Schafer and Schiller (2018) have also hypothesized that the mammalian neocortex contains maps of spatial objects as well as maps of social interactions.

Buzsaki and Moser (2013) consider cognitive maps in planning, an area the new work on the Causal Cognitive Architecture below develops. They propose that the memory and planning properties of the mammalian brain actually evolved from the same mechanisms used for navigation of the physical world. With regard to the neuroanatomical and neurophysiological basis for cognitive maps in the brain, work by Schuck and colleagues (2016) suggests that the human orbitofrontal cortex holds a cognitive map of the current states of a task being performed.

3 New Work: The Causal Cognitive Architecture 7 (CCA7)

3.1 Duplication of the TempMap Memory Areas

As noted in the Introduction section, the Causal Cognitive Architecture is a brain-inspired cognitive architecture (BICA) that was developed from the hypothesis that the navigation circuits in the amniotic ancestors of mammals duplicated many times to eventually form the neocortex. The thousands or millions (depending on the organism) of neocortical minicolumns are modeled in the

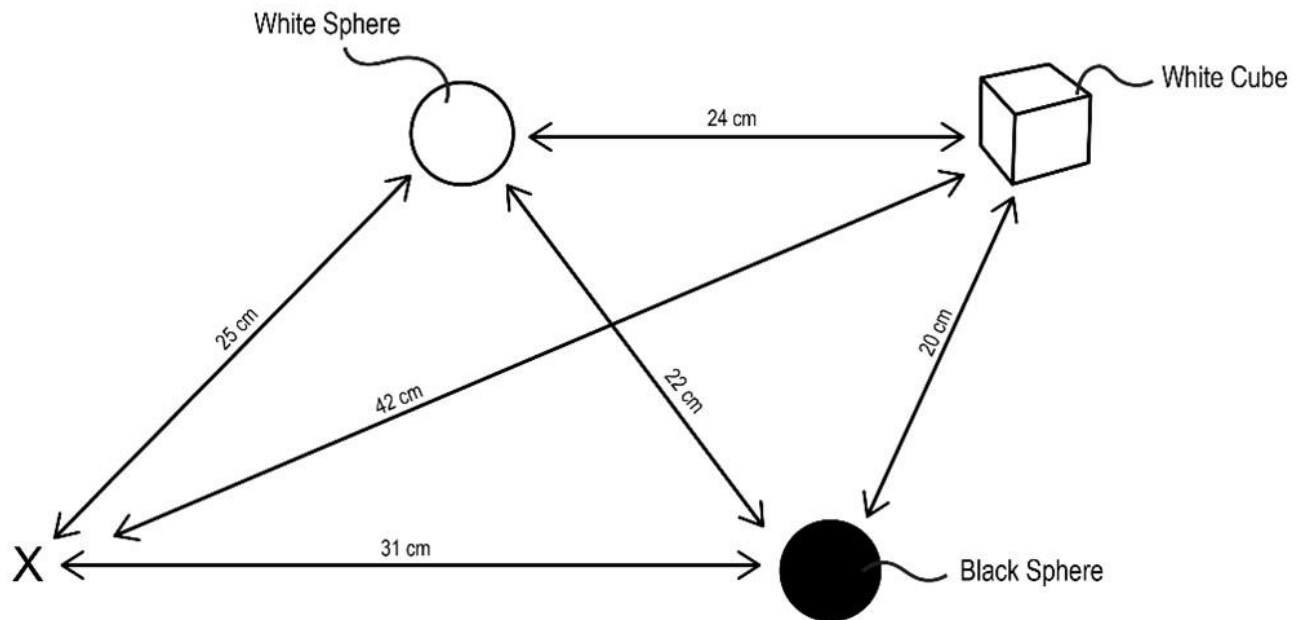


Figure 10. Starting at location “X”, the CCA7 robot must go to the white sphere, the black sphere and the white cube in any order, and then return back to the starting location. The instruction does not specify that the CCA7 should visit each object only once, but this will be implicit in the instinctive primitive triggered, as described in the text. Similarly, the instruction does not specify this, but implicit in the instinctive primitive triggered is that it should attempt to do this using the navigation path with the shortest distance.

air	air	air	air	air	air
air	air	air	air	air	air
air	air	air	air	air	air
air	air	24,22,25, sphere, white, link {0025,0,0,0}	24	20,42,24, block, white, link {0020,0,0,0}	air
air	25,42	22,42	42	20	air
25,42,31 cca7, link {0033,0,0, 0}	31	31	20,31,22, sphere, black, link {0024,0,0,0}	air	air

A. Navigation Module A

"back", link{+}					
"go", link{+}	"to", link{+}	"all", link{+}	"objects", link{+}	"and", link{+}	"go", link{+}

B. Navigation Module B n=1

Figure 11. **A.** While in location "X" the CCA7 robot maps the sensory scene of Figure 10 into the navigation map in Navigation Module A. The numbers refer to the distance (in centimeters) between the objects in the different cells. (The distance number can be determined by matching the same number in the path between two cells. As well, note a clockwork recording of distances in each cell.) **B.** The instruction "go to all objects and go back" is placed in Navigation Module B n=1 as shown

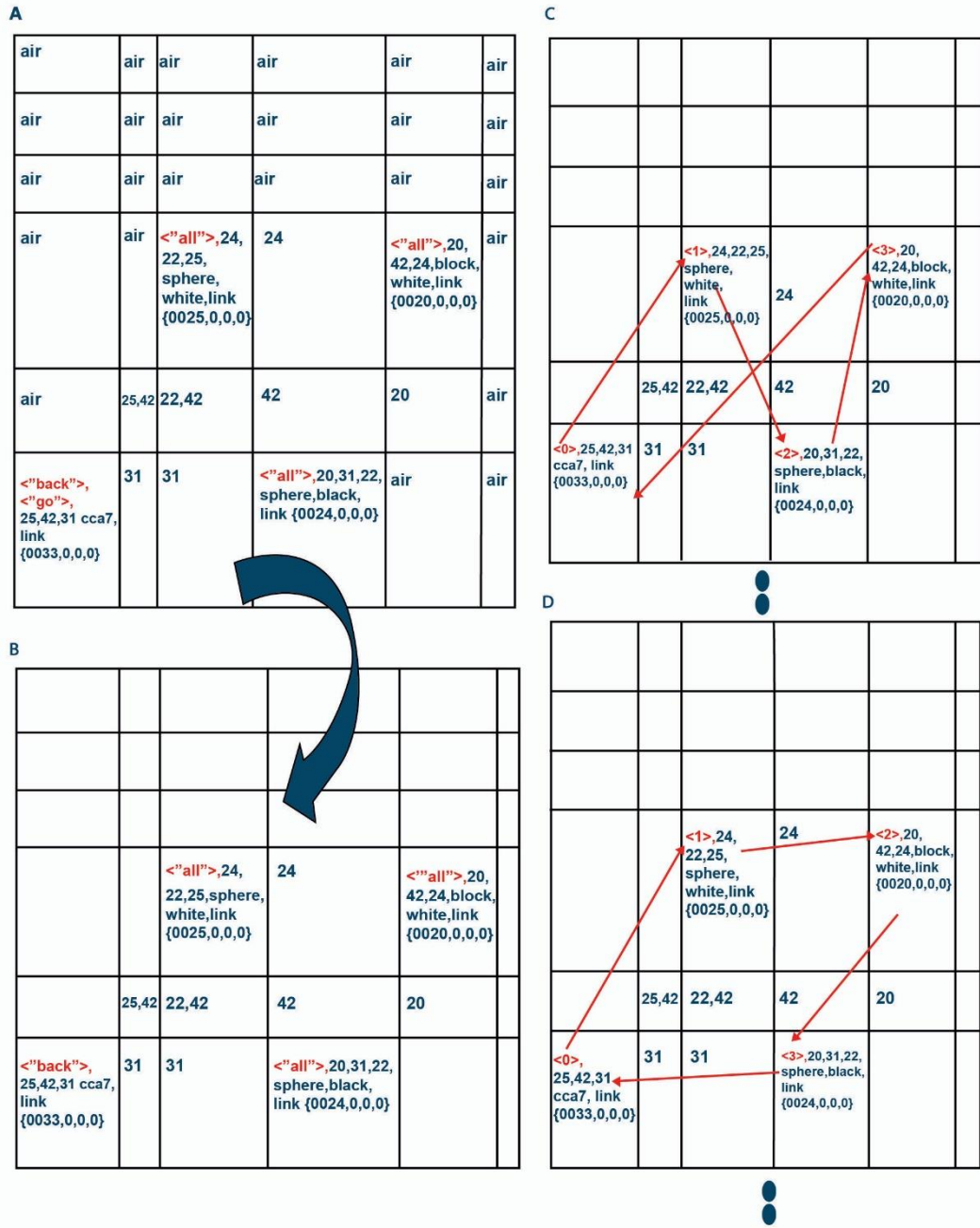


Figure 12. **A.** The navigation map in Navigation Module A is tagged via the operation of the “goto ()” instinctive primitive operating on the navigation maps in Navigation Module A and Navigation Module B.

B. The instinctive primitive “small_plan ()” copies Navigation Module A into the 1023 (n=1..1023) navigation maps in Navigation Module B’s, removing any action words and just leaving the tagged cells to navigate to. One of the navigation maps of Navigation Module B is shown here.

C. Navigation Module B n=1: The instinctive primitive “small_plan ()” uses as the starting

point the tagged cell and considers which distance to another tagged cell (i.e., representing an object) is the shortest. The cell that has the shortest distance is tagged with a <1>. Then it considers cell <1> as the starting point and considers which distance to another tagged cell is the shortest. This continues until all the tagged cells in the navigation map are considered and re-tagged with a number indicating in which order they should be navigated to. After all cells are navigated to, there is a return to starting position which has been tagged with a <0>. This simple nearest neighbor algorithm occurs in the navigation map in the n=1 Navigation Module B. Note that the sum of the distances is $25+22+20+42=109$ cm in this navigation map.

D. Navigation Module B n=2: The simple nearest neighbor algorithm occurs again in the navigation map in the n=2 Navigation Module B, however, random fluctuations have been introduced (see text). The instinctive primitive “`small_plan()`” considers the starting tagged cell and considers which distance to another tagged cell (i.e., representing an object) is the shortest. Random fluctuations are now introduced. Note that when deciding which cell to navigate to after cell (2,2,0) containing the white sphere, the random fluctuation introduced here causes “`small_plan()`” to choose cell (4,2,0) even though the distance of 24 to that cell was not the nearest neighbor. Note that the sum of the distances is $25+24+20+31=100$ cm in this navigation map.

architecture as navigation maps. The modeling of the mammalian brain and its evolution is done in a loosely functionalist approach (e.g., Lieto, 2021b) with constraints imposed by structuralist concerns. Small modifications in the architecture, akin to what could have been reasonable genetic and developmental changes, have been postulated and explored in the development of the versions of the architecture from the Causal Cognitive Architecture CCA1 version to the CCA6 version.

This very functionalist and theoretical approach to mammalian brain functioning and evolution is quite different than approaches that have attempted to more faithfully replicate brain structure and function (e.g., Markram, 2012; Frégnac, 2023). However, the approach taken by the Causal Cognitive Architecture does allow the emergence of mechanisms which could hypothetically explain the functioning of the mammalian brain as well as how ordinary genetic and developmental mechanisms could have readily allowed the emergence, i.e., evolution, of the seemingly discontinuous features in humans (i.e., the sharp cognitive and behavioral differences between humans and our closest evolutionary relatives). As well, the approach taken by the Causal Cognitive Architecture creates a mechanism (i.e., a particular cognitive architecture) that can be used as the basis of building intelligent artificial systems.

As noted above, in this paper, the question is asked that what if the evolution of the human brain were to continue as it has in the past, and given an environment selecting for the ability to better solve complex problems which humans encounter in their lives (very roughly indicated by measures of intelligence, for example, e.g., Legg and Hutter, 2007; Wang, 2019; Adams et al., 2012), then what advantageous changes could occur as reflected in a model such as the Causal Cognitive Architecture?

A computer engineer interested in enhancing the “intelligence” (as per the definitions above) capabilities of the CCA6 version of the architecture (Figure 5), could readily add a large language model (LLM) module to the architecture, or even just add a simple calculator module to the architecture. If one assumes that the CCA6 could be developed to the point of human intelligence (i.e., with adequate instinctive and learned primitives and with adequate experiences stored throughout the architecture), then adding even a calculator module could create a super-human

intelligence (albeit, given the assumptions above). For example, in computing various strategies or outcomes, numerical answers would be readily available for many problems in such an architecture, unlike in the CCA6 version shown in Figure 5 or unlike in the case of an actual human.

Adding a calculator module or even more advantageously adding a complete or multiple LLM modules to the Causal Cognitive Architecture in Figure 5 could be considered in future work. Indeed, adding LLMs to cognitive architectures is an active area of research at the time of writing (e.g., Joshi, Ustun, 2023; Laird et al., 2023; Sun, 2024). However, in this paper the assumption is that there will be an environment selecting for the ability to better solve complex problems. Thus, although there is not in this paper the construction and mutation and testing of millions of copies of the CCA6, there is a consideration of what advantageous modifications could emerge next, rather than design in modules (e.g., calculator module, LLM, and so on) which would not emerge naturally on their own as such. (A calculator module or a complex LLM module would not readily emerge on its own from the CCA6 version of the architecture shown in Figure 5.)

It is hypothesized that a first step in such an evolution could be the duplication of the **TempMap** temporary memory area in Navigation Module B into many such **TempMap** temporary memory areas. As noted above, various mechanisms are feasible for the duplication of brain structures (e.g., Rakic, 2009; Chakraborty and Jarvis, 2015). Thus, as a first step in the continued evolution of the CCA6 version of the architecture, there is the multiple duplication of the **TempMap** temporary memories in Navigation Module B. This is shown in Figure 8. Previously there was a single **TempMap** temporary memory area in the Navigation Module B; now there are many.

The temporary memory area **TempMap** was discussed above in its use for allowing analogical reasoning (97). Mammalian brain working memory, in particular human working memory, is the inspiration for the architecture's Navigation Modules and the **TempMap** temporary memory. There is in fact in the population variability in human working memory capacity. Work by Friedman and colleagues (2008) claims that individual differences in executive function and by implication human working memory are almost completely genetic in origin. However, despite decades of research on working memory, its measurement still remains uncertain in many regards (Cowan, 2001; Ma et al., 2014; Oberauer et al., 2016; Chuderski and Jastrzebski, 2018; Friedman and Miyake, 2017; Carruthers, 2013; Ando et al., 2001). Although only a single **TempMap** memory was required by the CCA5 or CCA6 versions of the architecture for compositional language processing (Schneider, 2024), it is known that in humans higher working memory capacity is associated with higher intellectual performance (e.g., Aubry and colleagues, 2021). As noted above, Navigation Module B is associated with compositional operations. Thus, the additional temporary memories incorporated into Navigation Module B, as shown in Figure 8, could allow more complex instinctive primitives and more complex learned primitives to emerge that require additional temporary memory storage. This will be explored below.

3.2 Duplication of Navigation Module B's

While it is possible to navigate by simple rules/heuristics or similarly generate words of a communication by simple rules/heuristics, planning a navigation route, or planning words to generate in a communication, or planning any other similar task, can be advantageous. In any task where planning can improve the outcome, there are usually many possible paths that can be chosen, and it can be very advantageous to run possible plans in parallel.

Thus, it is hypothesized that another step in the evolution of the Causal Cognitive Architecture could be the duplication of Navigation Module B into many such Navigation Module B's. As noted above, various mechanisms are feasible for the duplication of brain structures (e.g., Rakic, 2009; Chakraborty and Jarvis, 2015). Thus, as the next step in the continued evolution of the CCA6 version of the architecture, there is the multiple duplication of the Navigation Module B's. This is shown in Figure 9. The evolved architecture (i.e., multiple Navigation Module B's and multiple temporary memories within each of the Navigation Module B's) is named the Causal Cognitive Architecture 7 (CCA7).

In the CCA7 version of the architecture shown in Figure 9 there are 1024 copies of Navigation Module B. In every single Navigation Module B there are 1024 **TempMap** temporary memories (the number is not shown in the figure). The temporary memories are accessible for the operations of present and future instinctive primitives and learned primitives. Each **TempMap** temporary memory is capable of storing and representing a navigation map.

Consider the well-known traveling salesperson problem where a salesperson, or in this case an agent controlled by a CCA7 version of the architecture, must find the shortest route in order to visit only once each of a number of different cities and then return to the starting city. This is an NP-hard problem where the number of possible navigation routes to consider in finding the best solution increases exponentially with the number of cities. However, in the CCA7 version of the architecture, given that there are now approximately one thousand Navigation Module B's, then many of the possible routes (or promising routes given the exponential nature of the problem) can be evaluated in parallel, and a more optimal route planned ahead of time. This will be explored below in more detail, including a detailed examination of the CCA7 version of the architecture's internal operations and internal navigation maps.

3.3 The Traveling Salesperson Problem

As noted above, in the traveling salesperson, the salesperson (or in this case an agent controlled by a CCA7 version of the architecture) must find the shortest route in order to visit only once each of a number of different cities and then return to the starting city. The many Navigation Module B's should allow the CCA7 to evaluate many of the possible (or promising) routes in parallel, and plan a more optimal route ahead of time.

For example, if there are a half-dozen cities (or locations or other equivalent destinations) that need to be visited then this represents $(6-1)!$ or 120 navigation routes. (Actually only 60 of these routes need to be considered—returning home to the original city creates a cyclic graph which can be navigated forwards or backwards.) If each possible route can be represented in a separate navigation module and there are hundreds of navigation modules in the architecture available with each running a different combination of routes, then this problem can be solved much faster than if only a single navigation module was available.

If there were, for example, a dozen cities (or locations) that need to be visited, then this represents $(12-1)!/2$ or nearly twenty million navigation routes to explore in order to find the best solution. Even with a thousand navigation modules this would not be enough to run each possible navigation route in a separate navigation module. However, having the thousand navigation modules, in combination with other instinctive primitives and learned primitives of the architecture, can still greatly accelerate a reasonable solution in this case. For example, the nearest neighbour solution algorithm is a

relatively simple algorithm where the agent chooses the nearest city (or location) as the next city to visit (Rosenkrantz et al., 1977). However, this algorithm can sometimes give very poor solutions, i.e., very lengthy navigation routes to the problem (Bang-Jensen et al., 2004). However, by having over a thousand different navigation modules, it is possible to consider over a thousand different implementations of the nearest neighbor solution algorithm. Without any sophisticated algorithms (e.g., simply apply random choices for some cities rather than the nearest, e.g., simply apply various local properties such as avoiding crossings or not avoiding crossings of paths, etc.) by using the thousand plus navigation modules to run slightly different solutions to the problem, the architecture can better ensure that the solution produced is less likely to be one of the worst solutions.

There is a very large literature on solutions to the traveling salesperson problem. A myriad of algorithms have been proposed, including many parallel solutions (e.g., Tschoke et al., 1995). For example, Dorigo and Gambardella (1997) describe using an algorithm based on a colony of ants to finding successively shorter routes by laying down pheromone trails. Of interest, for certain variants of the problems, humans can visually produce solutions that are close to the optimal solution (Dry et al., 2006). While the literature gives much more sophisticated possible solutions, the traveling salesperson is considered here simply as an example to illustrate that having multiple navigation modules can be greatly advantageous to various planning strategies the architecture is required to perform.

3.4 `small_plan()` Instinctive Primitive

As discussed above, instinctive primitives are effectively small procedures operating on the contents of the navigation map(s) in the navigation module(s). The instinctive primitives are inspired by the work of Spelke and colleagues who have described many innate behaviors in human infants (Spelke, 1994; Spelke, Kinzler, 2007). Human infants do have innate behaviors with regard to simple planning (e.g., Liu et al., 2022; Claxton et al., 2003; McCormack and Atance, 2011). Thus, given the brain-inspired origins of the architecture, it is reasonable that the CCA7 architecture contains an instinctive primitive capable of simple planning (as opposed to learning how to do simple planning via a learned primitive). The CCA7 architecture now includes an additional instinctive primitive “`small_plan()`” for simple planning.

The instinctive primitive “`small_plan()`” can use a single Navigation Module B as in the case of the CCA6 version of the architecture (Figure 5), or in the case of the CCA7 it will make use of all the Navigation Module B’s present (which in Figure 9 consists of 1024 modules). The simultaneous usage of over a thousand navigation modules does not reflect, of course, a similar innate behavior described by Spelke and colleagues. The details and operation of “`small_plan()`” are discussed in the section below. (Of course, with education the CCA7 can acquire learned primitives that allow it better planning strategies, including better algorithms for the solution of the traveling salesperson problem. This is beyond the scope of this paper and is not discussed here.)

3.5 Operation of the Causal Cognitive Architecture 7 (CCA7)

Consider an agent, i.e., a robot, controlled by the CCA7 architecture shown in Figure 9. For simplicity the CCA7 architecture and the robot embodiment will be called the “CCA7” or “CCA7

robot.” The CCA7 robot comes to location “X” in Figure 10. It receives the instruction that starting at its existing position (i.e., “X”) it must visit each object and then return to the starting location.

While in location “X” the CCA7 robot maps a sensory scene into the navigation map in Navigation Module A, which is what it automatically does in each available cognitive cycle when there are new sensory inputs to process. The resulting navigation map in Navigation Module A is shown in Figure 11A. The CCA7 robot receives distances (either with the visual sensory information or via a separate ultrasonic distance sensory system). The numbers refer to the distance (in centimeters) between the objects in the different cells. (The distance number can be determined by matching the same number in the path between two cells. As well, note a clockwork recording of distances in each cell.) The instruction “go to all objects and go back” is placed in Navigation Module B as shown in Figure 11B. These operations are similar in nature to ones already described above for the CCA6 version of the architecture in its initial processing of the example of the sensory scene and instruction concerning the “placing a black sphere on top of the black block which is not near a cylinder” (Figures 6, 7).

However, as described above and shown in Figure 9, there are now in the CCA7 multiple Navigation Modules—one Navigation Module A and over a thousand (1024) duplicated Navigation Module B’s. Equation (115) (taken from Appendix A in the Supplementary Material section) indicates that the Working Navigation Map B’ **WNMB**’ (upon which primitives operate in Navigation Module B) is an array like before, but now can be one of 1024 different navigation maps (corresponding to a different navigation map in each of the Navigation Module B’s.)

The Navigation Module B’s are numbered $n=1$ to $n=1023$. The top (or first) Navigation Module B appears to be the $n=1$ Navigation Module B, as shown in Figure 9. However, a $n=0$ Navigation Module B exists and is used to store a copy of the compositional instructions so that if the other layers are overwritten, there is still a copy of the instructions. Layer $n=0$ is considered “reserved” and will not be overwritten. If there is other information that an instinctive or learned primitive needs to ensure remains intact for the current operations, other Navigation Module B’s can be temporarily designated “reserved” as well.

Equation (116) indicates that the same instinctive primitive or the same learned primitive (i.e., procedure) is initially applied to all of the Navigation Module B’s. As discussed below, random fluctuations can be introduced in the different Navigation Module B’s to produce a variety of results to choose from. In the example below (i.e., a traveling salesperson problem) the same instinctive primitives are used by all the Navigation Module B’s, and this does not cause any particular issues. However, in other types of problems, in subsequent cognitive cycles the initial primitive applied may trigger different primitives in different Navigation Modules. This issue is discussed below in the Discussion section.

At present, there is no energy-saving operation or Autonomic Module (Figure 9) interaction to turn off the multiple Navigation Module B’s and use only a sole Navigation Module B $n=0$ or $n=1$, i.e., much like the previous CCA6 version of the architecture functioned. However, if an instinctive primitive or learned primitive does not require the thousand-plus Navigation Module B’s it can simply ignore the results in the multiple modules, and use the results of operations in Navigation Module B $n=1$. This is also discussed below in the Discussion section.

As shown in Figure 11A is Navigation Module A containing Working Navigation Map (**WNMA**) of the sensory scene of the various places the agent has to navigate to. In Navigation Module B $n=1$

(Figure 11B) is a Working Navigation Map (**WNMB**_{n=1}) of the instruction sentence to “go to all the objects and go back.”

The word “go” in the first cell of the navigation map in Navigation Module B (Figure 11A) is matched against the Causal Memory Module as an action word and triggers the instinctive primitive “goto ()” (117). “**WNMA**_t = Nav_ModA.goto ()” indicates that this instinctive primitive, i.e., “goto ()” is being applied to the Working Navigation Map A in Navigation Module A.

The instinctive primitive “goto ()” causes the CCA7 robot to tag a location(s) and then essentially move to whatever location is indicated by the tag(s). The word “all” which is associated with active word <“go”> (until another action word is encountered, as in an earlier example above) will cause the tag <“all”> to be placed in all the cells with objects in the navigation map in Navigation Module A (Figure 12A).

The words “go back” are also associated with the instruction word <“go”> and will cause the tag <“back”> to be placed in the starting cell (which is (0,0,0) in this example). This can be seen in Figure 12A.

Once the instinctive primitive “goto ()” tags the cell(s) where it has to move to, it then decides if it will move (i.e., “go to”) the cell with the tag. However, if there are multiple tags, i.e., multiple locations to navigate to (“locations > 1” in (118–123)), then the “small_plan ()” instinctive primitive is activated instead of moving to a single location. As discussed above, this instinctive primitive will plan a navigation route to whatever multiple tagged locations are indicated on the navigation map(s).

Once activated (118), the instinctive primitive “small_plan ()” (regardless of its argument) will copy the Navigation Map A to all “non-reserved” Navigation Module B’s, i.e., n=1...1023 in this example. It will remove any action words such as “go” in the example above. This copying is indicated by the arrow in Figure 12. Thus, the instruction sentence in Navigation Module B n=1 (Figure 11B) is overwritten here. (Navigation Module B n=0 is “reserved” for a copy of the instruction sentence, although it actually will not be used again in this example.) A number of existing operations in various instinctive primitives have properties whereby they compare the contents of Navigation Module A and Navigation Module B with each other. Thus, the emergence of this step is a feasible one in the continued evolution of the architecture.

Equation (118) describes instinctive primitive “small_plan(random=False)” acting on the Working Navigation Map (**WNMB**_{t, n=1}) in Navigation Module B n=1 (Nav_ModB_{n=1}) at time t=t (i.e., during the current cognitive cycle). The instinctive primitive “small_plan(random=False)” follows the nearest neighbour algorithm discussed above. In making a plan where to navigate to, this primitive will choose the tag (i.e., location) that is closest to the tag (i.e., location) from where it is navigating. The argument random=False indicates that this instinctive primitive does not introduce any random variations. As will be seen below, in the other Navigation Module B’s n=2...1023 random variations will be introduced.

In the n=1 Navigation Module B, “small_plan(random=False)” operates on the navigation map shown in Figure 12B and determines which tagged cell to navigate first. It uses a nearest neighbor algorithm in its planning actions. For example, in Figure 12B, the cell (0,0,0) in which the CCA7 is starting from, has a distance of 25, 31, and 42 units (actually centimeters, but “small_plan ()” will disregard the actual units) to the other objects. (They are listed as “25, 42,

31” in cell (0,0,0) in Figure 12B due to a clockwise organization of distances.) According to the nearest neighbor algorithm it chooses the shortest distance, which is 25, i.e., it plans to navigate first to cell (2,2,0) containing the white sphere. Thus, it changes the <“all”> tag to a <1>. This can be seen in Figure 12C.

The instinctive primitive then considers navigating from cell (2,2,0)—which object to navigate to next? As can be seen in Figure 12B or Figure 12C, 22 is the shortest distance, thus it decides to navigate to cell (3,0,0) which contains the black sphere. It changes the <“all”> tag to a <2>. It then considers navigating from cell (3,0,0)—which object to navigate to next? Actually, the only untagged object remaining is the white block in cell (4,2,0), which is then tagged with a <3>. This is shown in Figure 12C.

If this was the previous CCA6 version of the architecture (albeit, retrofitted with these new equations) with only one Navigation Module B, then at this point the instinctive primitive “`small_plan(random=False)`” would trigger the instinctive primitive “`move()`” to move a CCA6 robot to cell (2,2,0) containing the white sphere. Then the instinctive primitive “`move()`” is triggered again to move to cell (3,0,0) containing the black sphere. Then the instinctive primitive “`move()`” is triggered again to move to cell (4,2,0) containing the white block. Then the instinctive primitive “`move()`” is triggered again to move to cell (0,0,0) which was the starting point. From Figure 12C, note that the sum of the distances is $25+22+20+42=109$ cm in this navigation route.

However, in the CCA7 version of the architecture being considered here, there are over a thousand Navigation Module B’s. As equation (120) indicates, for Navigation Module B $n=2\dots1023$ the instinctive primitive “`small_plan(random=weight_distance)`” will perform a similar nearest neighbor planning algorithm in the other modules for this same navigation map (Figure 12B). However, as indicated by the argument `random=weight_distance` random fluctuations are introduced now, so a slightly different navigation route may occur in different Navigation Module B’s $n=2\dots1023$ (14D).

The instinctive primitive “`small_plan(random=weight_distance)`” follows a similar nearest neighbor algorithm to the one described above. However, now random fluctuations may (or may not) be introduced at each step a navigation decision is made. These fluctuations are weighted by distance position, as explained below. Normally, the destination with the shortest distance will be chosen, as seen above for Navigation Module B $n=1$ (Figure 12C). Here this is likely to occur also, but some randomness means another destination can be chosen (e.g., Figure 12D), although the destinations the farthest away are the least likely to be chosen as the next destination, as will be shown below.

Consider that there is at any given decision point the list ***destination*** containing sorted destinations [a ,b, c, d, e...] which still can be navigated to (124). This list is sorted by distance such that navigation to destination “a” is the shortest, then navigation to destination “b” is the next shortest, and so on (125). The value a in the list is the distance to destination “a”, the value b in the list the distance to destination “b”, and so on.

Consider an example where there are 5 possible destinations which the CCA7 can now navigate to from some starting point, i.e., to object “a”, to object “b”, to object “c”, to object “d” or to object “e”. As per (124) ***destination*** = [a, b, c, d, e] , where the distance from the starting point to “a” is less than or equal to the distance from the starting point to “b”, and so on (125). Object “a” represented by element a in ***destination*** is considered to have position=1 in the list, while object “b” has position=2,

and so on. Similarly, object “a” is considered to have $\text{inverse_position}=5$ in this list, while object “b” has $\text{inverse_position}=4$, and so on (124).

Equation (126) shows that when the instinctive primitive “ $\text{small_plan}(\text{random}=\text{weight_distance})$ ” is triggered, a parameter “ weight ” is given a value of 4. Equation (127) shows that when the instinctive primitive “ $\text{small_plan}(\text{random}=\text{weight_distance})$ ” is triggered, the probability of selecting destination “x” to navigate to is given by “ $\text{probability_destination}_x$ ” which can be computed as “ $\text{inverse_position}_x^{\text{weight}} / \sum \text{inverse_position}^{\text{weight}}$ ”.

Continuing with the example above of choosing to navigation to locations “a”, “b”, “c”, “d” or “e”, consider equations (126) and (127). Consider navigating to the first destination “a” (which is the shortest navigation path from the starting point since it is the first element in **destination**). Thus, as per (127), $x = \text{“a”}$ and the value of the term “ $\text{inverse_position}_a^{\text{weight}}$ ” is thus 5^{weight} . The parameter weight is 4 (126), thus the value of the term “ $\text{inverse_position}_a^{\text{weight}}$ ” is 5^4 , or 625. Similarly, the value of all the inverse positions raised to the fourth power ($\text{weight}=4$) added up, i.e., “ $\sum \text{inverse_position}^{\text{weight}}$ ” is $625+256+81+16+1=979$ (127).

(In the actual CCA7 version of the architecture, other than as needed internally (and encapsulated) for artificial neural networks being used, only very simple arithmetic is explicitly available. Thus, in (127) the “ $\text{probability_destination}_x$ ” is shown as being approximately equal to a term which must be calculated via high exponential powers and involves the manipulation of many decimal places. While (127) is fine for some simulations of the architecture, the relationship shown in (127) can be achieved more realistically by the architecture by making use of stored probability distributions (see below). A limited number of such probability distributions can approximate (127) in deciding which object or city to navigate next to in a planning task.)

Continuing with the example above of choosing to navigation to locations “a”, “b”, “c”, “d” or “e”, the instinctive primitive “ $\text{small_plan}(\text{random}=\text{weight_distance})$ ” has just been triggered. Thus, weight is given a value of 4 (126). The probability of the algorithm in this instinctive primitive choosing, for example, destination “a” to navigate next to, is $\text{probability_destination}_a$. By (127) this is equal to “ $= \text{inverse_position}_a^{\text{weight}} / \sum \text{inverse_position}^{\text{weight}}$ ”. Above the term “ $\text{inverse_position}_a^{\text{weight}}$ ” was calculated to be 625, and the term “ $\sum \text{inverse_position}^{\text{weight}}$ ” to be 979. Thus, the probability of the algorithm in this instinctive primitive to choose destination “a” to navigate next to is $625/979$, or 64%.

From similar calculations the probability of choosing any of these sorted destinations (i.e., “a” is closer, “e” is the farthest away from the starting point) in this example of [a, b, c, d, e] is [64%, 26%, 8%, 2%, 0.1%]. Thus, when $\text{small_plan}(\text{random}=\text{weight_distance})$ is used in this example, of the five potential destinations to choose from in [a, b, c, d, e], there is in this example, a 64% chance of navigating to the nearest neighbor “a”, a 26% chance of navigating to the next nearest neighbor “b”, but only a 0.1% chance of navigating to the farthest neighbor “e”.

In (128) it can be seen that when the instinctive primitive “ $\text{small_plan}(\text{random}=\text{False})$ ” is triggered, a parameter “ weight ” is given a value of 30. The result of this high weight is that is that the nearest neighbor destination is always used, i.e., there is no randomness (129). Thus, the probability distribution for navigation to potential objects/cities [a, b, c, d, e] is [100%, 0, 0, 0, 0], i.e., there is a 100% chance of choosing object/location “a” to navigate to, and 0% chance of choosing object/location “b”, “c”, “d” or “e” to navigate to.

As noted above, if there is only one Navigation Module B in the system, or if this is a CCA7 version of the architecture and this is Navigation Module B $n=1$, then as equation (118) indicates, the instinctive primitive “`small_plan(random=False)`” is triggered. The nearest neighbor (i.e., shortest distance) from the starting point of cell (0,0,0) (Figure 12C; there is a <0> put in that cell) is cell (2,2,0)—there is a <1> put tag in that cell. The next nearest neighbor is cell (3,0,0)—there is a <2> put tag in that cell. The next nearest neighbor is the only one left which is cell (4,2,0)—there is a <3> tag put in that cell. Then with no more active cells to navigate to left, there is navigation back to the starting point of (0,0,0)—there is a <0> tag there.

Once all cells are tagged, the instinctive primitive “`small_plan(random=False)`” would trigger the instinctive primitive “`move()`” to move a CCA7 robot to the tagged cells (122, 123). The instinctive primitive “`move()`” first moves the CCA7 to cell (2,2,0) with the tag <1> containing the white sphere. Then the instinctive primitive “`move()`” is triggered again to move to cell (3,0,0) with the tag <2> containing the black sphere. Then the instinctive primitive “`move()`” is triggered again to move to cell (4,2,0) with the tag <3> containing the white block. Then the instinctive primitive “`move()`” is triggered again to move to cell (0,0,0) containing the tag <0> which was the starting point. From Figure 10 and Figure 12C, note that the sum of the distances is $25+22+20+42=109$ cm in this navigation route.

Now consider the Navigation Module B’s $n=2 \dots 1023$ in the CCA7 version of the architecture. In Figure 12D Navigation Module B $n=2$ is shown. As equation (120) indicates, the instinctive primitive “`small_plan(random=weight_distance)`” is triggered. As before the instinctive primitive “`small_plan()`” considers which possible destination it can navigate to will be the shortest, albeit now with a random fluctuation introduced. From the starting point of cell (0,0,0) the CCA7 can navigate next to cells (2,2,0), (3,0,0) or (4,2,0). From Figure 10 it can be seen these correspond to possible distances of (25cm, 31cm and 42cm).

As discussed above, “`small_plan(random=weight_distance)`” will introduce a random fluctuation in deciding which object/location to navigate to via equations (124–127). The sorted list [25, 31, 42] (124, 125). The sum of the inverse positions is $3^4+2^4+1^4$, or 98, and thus the probability distribution is [81/98, 16/98, 1/98], or [83%, 16%, 1%]. The likelihood of navigating to the first position destination of 25cm corresponding to cell (2,2,0) is 83%, while the probability of navigating to (3,0,0) is 16% and the probability of navigating to (4,2,0) is 1%. A cumulative probability distribution results essentially from considering these probabilities: [$\leq 83\%$, $\leq 99\%$, $\leq 100\%$]. A random number between 0 and 1 is obtained which happens to be, for example, .55 or 55%. It is within the 83% cumulative probability corresponding to the first position destination. Thus, the CCA7 architecture tags cell (2,2,0) with a <1>.

Now the instinctive primitive “`small_plan(random=weight_distance)`” must consider navigating to the next object/location. From the starting point of cell (2,2,0) the CCA7 can navigate next to cells (3,0,0) or (4,2,0). From Figure 10 it can be seen that these correspond to possible distances of (22cm and 24cm). The sum of the inverse positions is 2^4+1^4 , or 17, and thus the probability distribution is [16/17, 1/17], or [94%, 6%]. The likelihood of navigating to the first position destination of 22cm corresponding to cell (3,0,0) is 94%, while the probability of navigating to (4,2,0) is 6%. A cumulative probability distribution results essentially from considering these probabilities: [$\leq 94\%$, $\leq 100\%$]. A random number between 0 and 1 is obtained which happens to be, in this case, for example, .95 or 95%. Thus, instead of navigating to the nearest neighbor (3,0,0), the CCA7 will tag the second nearest neighbor, i.e., (4,2,0) as the next destination to navigate to with

a <2> (Figure 12D). The next nearest neighbor is the only one left which is cell (3,0,0)—there is a <3> tag put in that cell. Then with no more active cells to navigate to left, there is navigation back to the starting point of (0,0,0)—there is a <0> tag there. The tagged cells can be seen in Figure 12D.

From Figure 10 and Figure 12D note that the sum of the distances is 25+24+20+31= 100 cm in this navigation route. Thus, even though this route ended up taking a path between two locations which was not the shortest distance (i.e., going from cell (2,2,0) to cell (4,2,0) which was 24cm rather than going to cell (3,0,0) which was 22cm) it turned out that the total distance in navigating to all object/locations turned out to be shorter than the path obtained in Navigation Module B n=1 where the nearest neighbor algorithm was followed at each decision point.

Similar algorithms are also running in the other Navigation Module B's at the same time. The total distance sum obtained in each Navigation Module B is transferred to the **TempMap** memory areas of Navigation Module B n=0 (119, 121). Although there are many more **TempMap** memory areas now available in the CCA7 version of the architecture, this instinctive primitive actually just keeps track of the navigation map number (i.e., which “n” from n=1...1023) which has yielded the smallest total distance of the best (i.e., shortest) navigation plan found (122). Thus **Nav_ModB_{n=best}** where “best” is the Navigation Module B “n” which showed the shortest total navigation distance.

The instinctive primitive “small_plan()” then activates the instinctive primitive **move(WNMB'_{t, n=best})** (123). To continue the above example, “best” is Navigation Module B n=2, i.e., shown in Figure 12D. (The total distance here was 100cm versus 109cm in Navigation Module B n=1, and versus 119cm obtained in other Navigation Module B's). It will then repeatedly trigger the instinctive primitive “move()” to navigate to <1> (the white sphere), then <2> (the white cube), then <3> (the black sphere), and then return to the starting cell <0>.

This navigation planning example involves navigating to three locations and then returning back to the starting position. Thus, there are only a handful of possible variations in navigation to consider, and thus, despite the random fluctuations, many variations will repeat among the over thousand Navigation Module B's. However, many real-world problems may involve more locations (or social situations or other analogous “locations”) than this simple problem. In the next section, the CCA7 architecture will be applied to a larger dataset of navigation locations.

The instinctive primitive “small_plan()” effectively helps to decide what sequence to perform operations in. While such problems can be physically moving to different locations, they can also range from navigating in the social hierarchy space of a society to involve navigating through an idea space of more abstract concepts. With regard to the traveling salesperson problem, or other particular planning problems, note that with education (i.e., assumes a larger set of instinctive primitives than exist at present, and acquisition of more basic concepts) the CCA7 can acquire learned primitives that are more specific and more sophisticated for particular planning purposes.

$$\mathbf{WNMB}'_{n=0...1023} = \in \mathbb{R}^{m \times n \times o \times p} \quad (115)$$

$$\begin{aligned} &(\mathbf{WNMB}'_{n=x} \neq \text{reserved} \text{ AND } \mathbf{WNMB}'_{n=y} \neq \text{reserved}) \\ &\Rightarrow \text{initial_primitive}_{\mathbf{WNMB}'_{n=x}} = \text{initial_primitive}_{\mathbf{WNMB}'_{n=y}} \quad (116) \end{aligned}$$

<“go”>

$\{\Rightarrow \mathbf{WNMA}'_t = \text{Nav_ModA.goto}() \text{ (117)}$

locations > 1,

$[\Rightarrow \mathbf{WNMB}'_{t, n=1} = \text{Nav_ModB}_{n=1}.\text{small_plan}(\text{random}=\text{False}) \text{ (118)}$

$(\Rightarrow \mathbf{TempMap}_{\mathbf{WNMB}'_{n=0}} = \text{minimum}(\text{total_distance}_{n=1}) \text{ (119)}$

$\Rightarrow \mathbf{WNMB}'_{t, n=2\dots1023}$

$= \text{Nav_ModB}_{t, n=2\dots1023}.\text{small_plan}(\text{random}=\text{weight_distance}) \text{ (120)}$

$\Rightarrow \mathbf{TempMap}_{\mathbf{WNMB}'_{n=0}} = \text{minimum}(\text{total_distance}_{n=2\dots1023}) \text{ (121)}$

$\Rightarrow \text{best} = \mathbf{TempMap}_{\mathbf{WNMB}'_{n=0, \text{minimum}(\text{total_distance})}} \text{ (122)}$

$\Rightarrow \text{Nav_ModB}_{n=\text{best}}.\text{move}(\mathbf{WNMB}'_{t, n=\text{best}})] \text{ (123)}$

$\mathbf{destination} = [\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e} \dots] \text{ (124)}$

$\text{distance}_a \leq \text{distance}_b, \text{distance}_b \leq \text{distance}_c, \dots \text{ (125)}$

$\text{small_plan}(\text{random}=\text{weight_distance}),$

$\{\Rightarrow \text{weight} = 4 \text{ (126)}$

$\Rightarrow \text{probability_destination}_x \approx \text{inverse_position}_x^{\text{weight}} / \sum \text{inverse_position}^{\text{weight}} \text{ (127)}$

$\text{small_plan}(\text{random}=\text{False}),$

$\{\Rightarrow \text{weight} = 30 \text{ (128)}$

$\Rightarrow \text{probability_destination}_{a, \text{weight}>9} = 1 \text{ (129)}$

4 Methods

4.1 Computer Simulation of the Causal Cognitive Architecture 7 (CCA7)

The equations (1) to (129) (Supplementary Material, Appendix A) are computer simulated via the Python language to represent the CCA7. The computer simulation does not interface with a real-time actual video camera or microphone inputs, or with real robotic actuators. Sensory inputs are simulated in all simulations, and actuator outputs are similarly simulated.

The navigation maps in the Python simulation have 6x6x0 dimensions (although internally a larger number of dimensions are actually used to represent segmentation of objects and binding with motion and action). As noted above navigation maps are essentially arrays. Thus, the more efficient Numpy library (Harris et al., 2020) is called by the Python program for most operations on the navigation maps. For future larger simulations of the architecture, more classical deep learning software and hardware can be used. However, in the current simulation the FuzzyWuzzy string matching library (via pypi.org) is used for pattern matching.

The Python simulation of the architecture at this time contains a very limited set of instinctive primitives. It mainly contains the ones specified in equations (1) to (129) which relate to very basic operations and the ability for causal reasoning, analogical induction, compositionality and as discussed above in the section on new work, simple planning. At this time, instinctive primitives must be hand crafted. Automated methods for instinctive primitive creation are being explored.

Python version 3.11 is used. The parallel elements of equations (1) to (129) are simulated sequentially—a new cognitive cycle starts when all the operations of the previous cognitive cycle have completed.

The main purpose of this computer simulation is to show that the operation of the CCA7 version of the architecture is feasible, particularly its ability to perform planning operations. The simulation, i.e., based on the representation of the CCA7 version of the architecture via equations (1) to (129), can be tested below on a traveling salesperson dataset. The distances between a starting city and a dozen other cities are given in Table 1 (Appendix B, Supplementary Materials) (Google OR-Tools, 2023). The results are discussed below.

4.2 Alternative Weightings for the Probability Distribution of the Next Destination

The literature on the traveling salesperson problem is vast and there are many strategies towards choosing the next location to navigate to (e.g., as mentioned above— Tschoke et al., 1995; Dorigo and Gambardella, 1997; Dry et al., 2006). Both strategic decisions and random fluctuations can be introduced into the solution algorithm in many ways. As noted above, the traveling salesperson problem is considered here simply as an example to illustrate that having multiple navigation modules can be greatly advantageous to various planning strategies the architecture is required to perform. Nonetheless, it is useful to consider how random fluctuations are inserted into the planning decisions. Of interest is that the positional weighting used in (124–127) does not take into account the relative values of the different distances, e.g., [12, 13, 44] will be weighted the same as [12, 42, 44], i.e., the probability of the choosing the location that is 42 units away will be the same as choosing the location that is 13 units away.

The reason for using the positional weighting is because the actual CCA7 version of the architecture only explicitly has access to very simple arithmetic. Thus, in weighting the probability distributions it was desired to use pre-stored distributions which could readily be accessed rather than involve complex calculations. Although (124–127) are used in the Python simulation of the architecture (albeit, necessitating the Python “Decimal” class due to the many digits created by the high-power exponents), the architecture can simply access a limited number of probability distributions based on the positions of the nearest city/location/object in a list, with no complex arithmetical calculations required.

It is possible, of course, to weight by the relative values: calculate the reciprocals of the difference of each number (nominal value of 1) in the list from the smallest value (thus, smaller difference will give a larger reciprocal) and normalize as probabilities. For example, if there are three possible locations to navigate to with distances (arbitrary units not specified) of [12, 13, 44] then the weight probabilities would be `normalize([1/1, 1/1, 1/32])`, or [49%, 49%, 2%]. Thus, there would be a 49% chance of navigating to the location 12 units away versus a 2% chance of navigating to the location 44 units away. The probability distribution for the example above of three possible locations [12, 42, 44] is `normalize([1/1, 1/30, 1/32])`, or [94%, 3.1%, 2.9%]. In contrast, the positional

probability distribution (i.e., via (124–127)) of either example yields [83%, 16%, 1%], i.e., 83% chance of navigating to the first-closest location, 16% chance of navigating to the second-closest location and a 1% chance of navigating to the third-closest location, regardless of the actual distances.

In a modified computer simulation, random fluctuations are introduced by comparing a normalized random number with the probability distributions calculated via relative weights as discussed above:

- a. calculate the reciprocals of the difference of each number (nominal value of 1) in the list from the smallest value;
- b. normalize the probabilities.

This alternative “value weighted” version can also be tested below on a traveling salesperson dataset. The distances between a starting city and a dozen other cities are given in Table 1 (Appendix B, Supplementary Materials) (Google OR-Tools, 2023). The results are discussed below.

4.3 Comparative Experiments

As noted above, the CCA7 architecture, functionally-based on a possible further evolution of the brain as modeled by previous versions of the Causal Cognitive Architecture, is expected to be able to perform simple planning in terms of navigating to multiple locations with a certain degree of higher efficiency than if the previous versions of the architecture were used.

Computer simulations of the CCA7 architecture can be tested on a traveling salesperson dataset. The distances between a starting city and a dozen other cities are given in Table B1 (Appendix B, Supplementary Materials) (Google OR-Tools, 2023).

The optimal (i.e., shortest) solution obtained via brute force (non-CCA7) computation is 7293 miles. (The data in Table 1 of distances between the cities was given in miles.) The route giving this shortest path is City #0,7,2,3,4,12,6,8,1,11,10,5,9,0 .

The traveling salesperson problem city data from Table B1 (Appendix B2, Supplementary Material) was simulated as sensory input data to the CCA7 architectures. The same CCA7 version of the architecture shown in Figure 9 was used. However, equations (126–129) were modified in different runs of the architecture as described below. As described above, the CCA7 architecture via the instinctive primitives associated with equations (126–129) attempts to produce the shortest path in a planning problem. In the case of the city data from Table B1, the architecture attempts to produce the shortest path to navigate once to the dozen cities listed in Table B1 and return back to the starting city (i.e., 13 cities in total).

The following questions were asked, and the accompanying comparative experiments were then performed:

- a. The position-weighted algorithm used to inject random fluctuations into the nearest neighbor algorithm (126–129) uses a *weight* parameter to create a probability distribution used in selecting the next destination to navigate to. At present a *weight* value of 4 is used.

What is the effect of varying the *weight* parameter on the shortest path yielded in the traveling salesperson problem, i.e., is the value of *weight* used in equation (126) a reasonable one, based on a typical planning problem represented by the data in Table B1?

b. Do the multiple Navigation Module B's allow better planning in terms of this traveling salesperson problem represented by the data in Table B1?

Multiple runs to ensure statistical significance (or insignificance) of the following is to be examined: the shortest distance obtained by a CCA7 architecture modified to use only 1 Navigation Module B versus a CCA7 architecture using 1023 Navigation Module B's.

Note: When multiple Navigation Module B's are used, Navigation Module B n=0 is restricted for holding a copy of any instructions, hence, 1K-1 results in 1023 Navigation Module B's available.

Note: As per equation (118) Navigation Module B n=1 uses the instinctive primitive `small_plan(random=False)` thus *weight* is set to 30 for this case, i.e., nearest-neighbor algorithm without any random fluctuations. However, the Navigation Module B's n=2...1023 per equation (120) use the instinctive primitive `small_plan(random=weight_distance)` thus *weight* is set to 4 for these Navigation Modules and there will thus be the possibility of random fluctuations injected in choosing the next destination city at every decision point.

Note: Due to the generation of high exponents and large decimal arithmetic seen in equation (127), when the *weight* parameter exceeds 9, as per equation (129) the probability of choosing the shortest distance becomes 100%, i.e., nearest-neighbor algorithm without any random fluctuations is used.

c. Do higher quantities of Navigation Module B's result in significantly better results?

Multiple runs to ensure statistical significance (or insignificance) of the following is to be examined: the shortest distance obtained by a CCA7 architecture using 1023 (i.e., "1K") Navigation Module B's versus versions of the architecture using 4,095 (i.e., "4K") and 16,383 (i.e., "16K") Navigation Module B's.

Note: Navigation Module B n=0 is restricted for holding a copy of any instructions, and thus the availability of 1K-1, 4K-1 and 16K-1 Navigation Module B's, which are rounded and simply referred to as "1K", "4K" or "16K" respectively.

d. Does a "value weighted" algorithm to introduce random fluctuations (i.e., Section 4.2) give better results (i.e., a shorter distance) than the "position weighted" algorithm (i.e., equations (124) to (127))?

Multiple runs to ensure statistical significance (or insignificance) of the following is to be examined: the shortest distance obtained by the usual "position weighted" version of the CCA7 architecture using 1023 Navigation Module B's versus the shortest distance obtained by "value weighted" version of the architecture.

Note: "Position weighted" refers to equations (124) to (127) which create a probability distribution of which location to choose next in deciding where to navigate to, simply in terms of the which location is the nearest, the next-nearest, the third-next-nearest, and so on, without considering the actual values (i.e., there is a relative ranking but without consideration of any scale) of the distances. In contrast, "valued weighted" refers to the modifications of these equations given by Section 4.2 such

that the reciprocals of the difference of each distance value from the smallest value (i.e., its actual value and the actual value of the smallest distance, rather than just its position of where it is relative to the other possible destinations) are calculated and normalized to create a probability distribution of which location to choose next in deciding where to navigate to.

5 Results

5.1 Effect of Varying the *weight* Parameter

As noted above, the position-weighted algorithm creates random fluctuations into the nearest neighbor algorithm (126–129). Lower *weight* values (equation (126)) make it less likely that the closest next destination will be navigated to, i.e., more likely another destination will be selected. At present a *weight* value of 4 is used. In order to see if this value is reasonable in terms of a typical planning problem as represented by the data in Table B1, the shortest distance obtained by the CCA7 for different values of the *weight* parameter were examined.

Table B2 (Appendix B, Supplementary Material) shows the shortest distance obtained by single runs of the CCA7 architecture for *weight* values varying from 1 to 30. The results shown in the column on the right side are for a version of the architecture with the full set of Navigation Module B's $n=1 \dots 1023$. For comparison, in the column on the left, only a single Navigation Module B is used, so rather than restrict it to equations (118, 119) which will yield each time a value of 8131 miles per the nearest-neighbor algorithm, it is also running equations (120, 121), i.e., `small_plan (random=weight_distance)`.

The data from Table B2's left and right columns are plotted respectively in Figure 13A and Figure 13B. As can be seen from these figures, the lower values of the *weight* parameter in the single Navigation Module B variant of the architecture give significantly poorer results than in the multiple Navigation Module B's version (i.e., what is shown in Figure 9) of the architecture.

In the multiple Navigation Module B variant of the architecture (Figure 13B) it can be seen that *weights* of 1 and 2 give poorer results than the *weights* of between 3 to 9. (*Weights* over 9 automatically cause the nearest neighbor algorithm to be used (128, 129).)

5.2 Multiple Navigation Module B's versus Original Architecture (Single Nav Module B)

Multiple runs to ensure statistical significance (or insignificance) of the data were done. As shown in Table B3 (Appendix B, Supplementary Material) 100 runs were observed for each variant of the architecture considered. The traveling salesperson problem data of Table B1 was used.

In Table B3 in the middle column are the shortest distances obtained with a version of the architecture using a single Navigation Module B utilizing a nearest neighbor algorithm. There is no randomness involved here and as can be seen, a shortest distance of 8131 miles was obtained in each of the 100 runs.

In Table B3 in the right column are the shortest distances obtained with a version of the architecture as shown in Figure 9 using 1024 Navigation Module B's. Navigation Module B $n=0$ is reserved. Navigation Module $n=1$ is used but set to use the nearest neighbor algorithm without randomness.

Navigation Module B's $n=2...1023$ use the nearest neighbor algorithm with the injection of random fluctuations with *weight* = 4 as per equations (120) to (127). The mean shortest distance obtained was 7432.2 miles with a standard deviation of 141.8 miles. 1% of the runs yielded the shortest distance possible of 7293 miles.

In table B3 in the left column are the shortest distances obtained with a version of the architecture using a single Navigation Module B but with the injection of random fluctuations with a *weight* parameter value of 4 as per equations (120) to (127). It does not reflect the CCA6 or the CCA7 architectures but was obtained to see that if running the single Navigation Module B variant (i.e., more similar to the CCA6 architecture) with random fluctuations would give better results. Note that the results show that the mean shortest distance was 9965.5 miles with a standard deviation of 1532.5 miles, i.e., worse results than obtained with the multiple Navigation Module B's variant shown in the right column. No runs yielded the optimal path of 7293 miles with the shortest distance of the 100 runs being 7647 miles.

The three columns of Table B3 are plotted in Figure 14. The green dots represent a version of the architecture using a single Navigation Module B but with the injection of random fluctuations with a *weight* parameter value of 4 as per equations (120) to (127). As can be seen from the figure, this variant of the architecture gives the largest distances, i.e., the worst results.

In Figure 14 the constant line of orange dots represents a version of the architecture using a single Navigation Module B following the nearest-neighbor algorithm without any randomness injected. The blue dots represent the CCA7 version of the architecture shown in Figure 9, i.e., multiple Navigation Module B's. As can be seen from this figure, these blue dots tend to give shorter distances, i.e., better results, than the orange dots.

To test for statistical significance between the results produced by the multiple Navigation Module B's version of the architecture (right-most column of Table B3) versus the single Navigation Module B running the nearest neighbor algorithm (middle column of Table B3), the probability of test results occurring due to random chance were calculated. A Welch's, one-tailed t-test was performed yielding a p value < 0.001.

Statistical significance of the results produced by the multiple Navigation Module B's version of the architecture (right-most column of Table B3) versus the single Navigation Module B running the similar position-weighted algorithm (left-most column of Table B3) was calculated via Welch's, one-tailed t-test. Again, a small p value was obtained and is recorded as $p < 0.001$ in Table B3.

5.3 1K vs 4K vs 16K Navigation Module B's

The CCA7 version of the architecture (Figure 9) was arbitrarily allowed 1024 Navigation Module B's in the emergence of the duplication of navigation modules. However, any other number is possible, albeit with neurophysiological considerations in the case of a biological brain or engineering considerations in the case of an artificial implementation of the architecture.

Table B4 compares 100 runs of the CCA7 version of the architecture operating on traveling salesperson problem data of Table B1. All versions are using a position-weighted method to inject randomness in the nearest neighbor algorithm, as per equations (118) to (129). While in the left-most column of Table B4 the architecture shown in Figure 9 and represented by equations (118) to (129) is

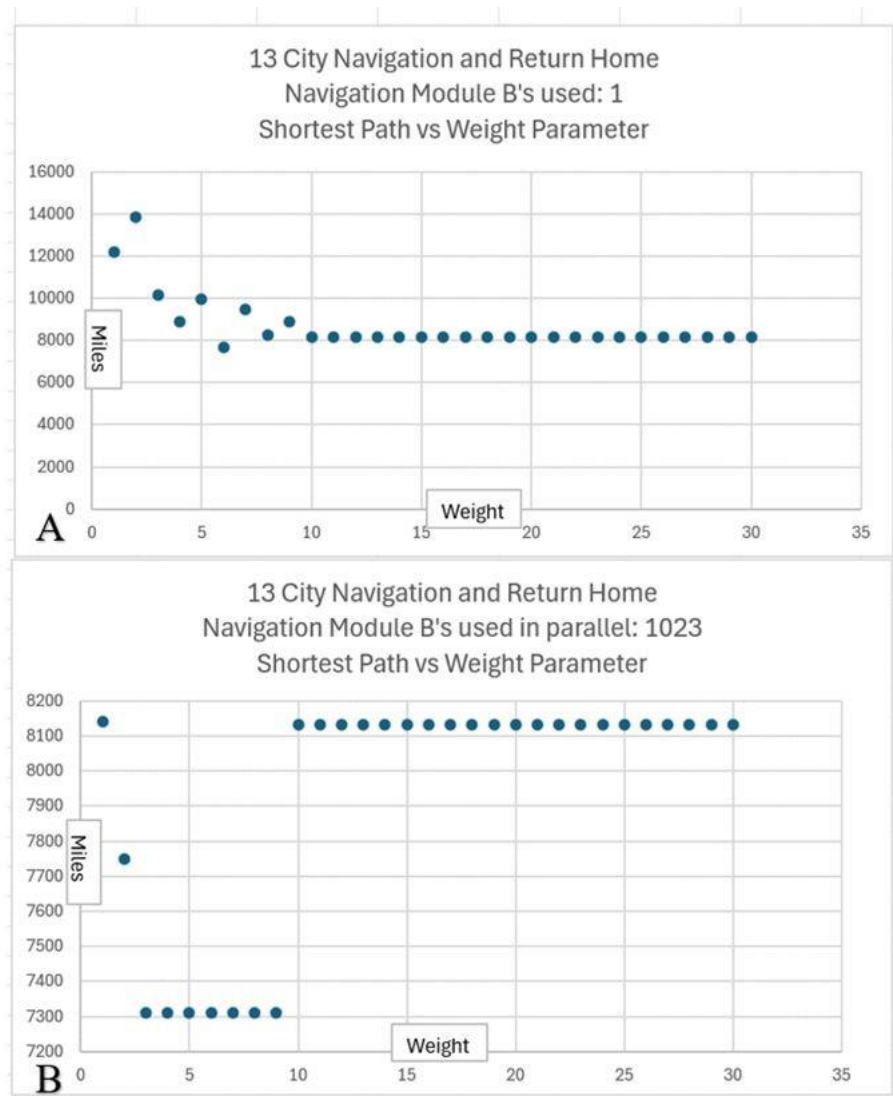


Figure 13. Simulation of the CCA7 robot navigating from a home city to twelve other cities (13 cities in total) and then returning home (Table B1, Supplementary Material). Lower values of *weight* introduce more randomness in choosing the next destination city (127).

A. Only a single Navigation Module B is used, so rather than restrict it to equation (118) (which will yield each time a value of 8131 miles per the nearest-neighbor algorithm) it is running equation (120), i.e., `small_plan(random=weight_distance)`.

B. The results are shown when the full set of Navigation Module B's $n=1\ldots 1023$ are used. Here too, all modules use `small_plan(random=weight_distance)` but note that at higher values of the *weight* parameter, the CCA7 follows the nearest neighbor algorithm.

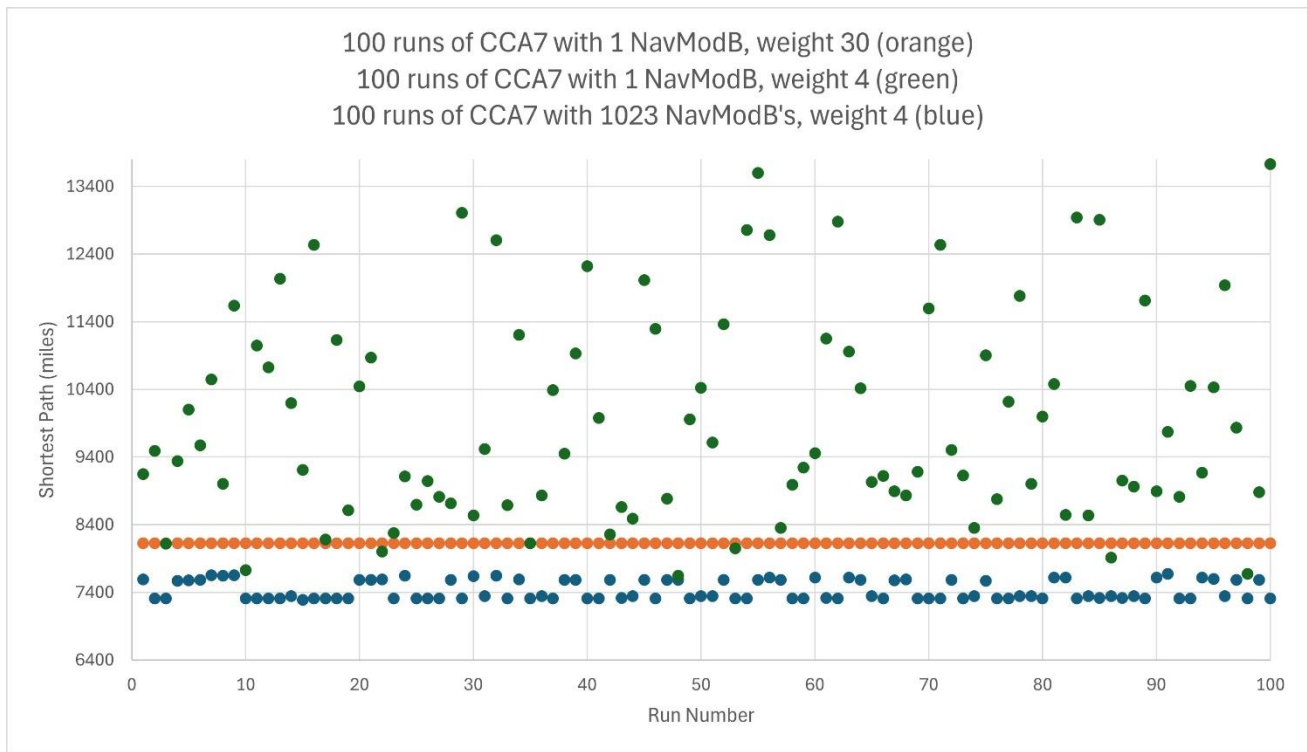


Figure 14. Simulation of the CCA7 robot navigating from a home city to twelve other cities (13 cities in total) and then returning home. 100 simulated runs with the same dataset (Table B1, Supplementary Material). Comparison of architecture with one Navigation Module B (with and without random fluctuations) versus with 1023 Navigation Module B's.

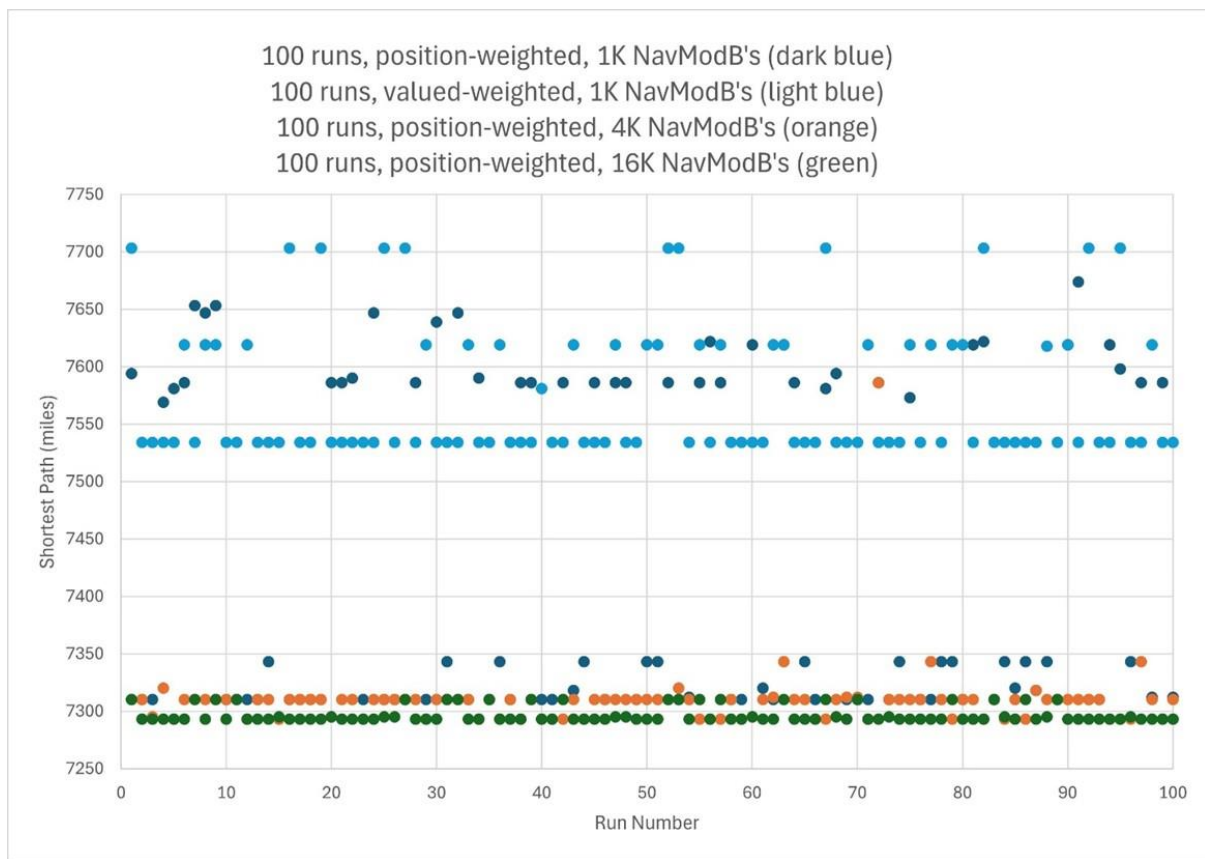


Figure 15. Simulation of the CCA7 robot navigating from a home city to twelve other cities (13 cities in total) and then returning home. 100 simulated runs with the same dataset (Table B1, Supplementary Material). Comparison of architecture with 1K, 4K and 16K Navigation Module B's, and comparison with 1K version running value-weighted algorithm (light blue dots).

run, in the middle column is a similar architecture except for the utilization of 4,095 Navigation Module B's, and in the right-most column is a similar architecture except for the utilization of 16,383 Navigation Module B's.

(Navigation Module B $n=0$ is restricted for holding a copy of any instructions, and thus the availability of 1K-1, 4K-1 and 16K-1 Navigation Module B's, which are rounded and simply referred to as "1K", "4K" or "16K" respectively.)

The CCA7 version with 4K Navigation Module B's has a mean shortest distance of 7309.1 miles (standard deviation 29.8 miles) versus 7432.2 miles (standard deviation 141.8 miles) for the 1K version. 27% of the runs of the 4K version yielded an optimal shortest distance path of 7293 miles versus only 1% of the runs of the 1K version. As shown in Table B4, Welch's 1-tail t-test was applied to this data and shows it is statistically significant at $p < .001$.

The CCA7 version with 16K Navigation Module B's has a mean shortest distance of 7296.8 miles (standard deviation 6.9 miles) versus 7309.1 miles (standard deviation 29.8 miles) for the 4K version. 67% of the runs of the 16K version yielded an optimal shortest distance path of 7293 miles versus

27% of the runs of the 4K version. As shown in Table B4, Welch's 1-tail t-test was applied to this data and shows it is statistically significant at $p < .001$.

The three columns of Table B4 are plotted in Figure 15. The dark blue dots represent the 1K version of the architecture, the orange dots represent the 4K version of the architecture and the green dots represent the 16K version of the architecture. As can be seen from this figure, while the 4K and 16K distances are close together, albeit with the 16K distances slightly smaller (although statistically significant as noted above), they are significantly smaller than the results produced by the 1K version of the architecture.

5.4 Position-Weighted Algorithm versus Value-Weighted Algorithm

As noted above, "position weighted" refers to equations (124) to (127) which create a probability distribution of which location to choose next in deciding where to navigate to, simply in terms of the which location is the nearest, the next-nearest, the third-next-nearest, and so on, without considering the absolute or relative values of the distances. Also as noted above, "valued weighted" refers to the modifications of these equations given by Section 4.2 such that the reciprocals of the difference of each distance value from the smallest value are calculated and normalized to create a probability distribution of which location to choose next in deciding where to navigate to.

Table B5 (Appendix B, Supplementary Material) compares 100 runs of the CCA7 version of the architecture operating on traveling salesperson problem data of Table B1. The column on the left represents the usual CCA7 version of the architecture position-weighted method to inject randomness in the nearest neighbor algorithm, as per equations (118) to (129). However, the column on the right represents a version of the CCA7 architecture using a valued weighted method to decide the probability of which location to navigate to at each decision point. Both architectures are using 1023 Navigation Module B's.

As shown in Table B5, the usual position weighted CCA7 version yields a mean shortest-distance of 7432.2 miles (standard deviation 141.8 miles) versus 7572.6 miles (standard deviation 58.0 miles) for the version using the valued-weighted method. The shortest distance (i.e., among the 100 runs performed) was 7293 miles for the position weighted CCA7 version versus 7534 miles for the value-weighted version of the architecture. 1% of the runs of the position-weighted architecture yielded the optimal shortest distance of 7293 miles while 0% of the value-weighted architecture runs yielded an optimal shortest distance. A Welch's, one-tailed t-test was performed yielding a p value < 0.001 .

The two columns of Table B5 are also plotted in Figure 15. The dark blue dots represent the 1K version of the position-weighted version of the architecture while the light blue dots represent the 1K version of the value-weighted version of the architecture.

6 Discussion

6.1 Interpretation of the Results

In the CCA7 architecture using 1023 Navigation Module B's, Figure 13B shows that a weight value between 3 to 9 gives the shortest paths, i.e., the best results. Thus, in equation (126) setting the

weight value to 4 appears reasonable for use with the further experiments done in this paper. Figure 13A was included for comparison, and it shows that in an architecture using only one Navigation Module B, the lowest weight values also give even poorer results.

Table B3 and Figure 14 compare a CCA7 version of the architecture using a single Navigation Module B with a CCA7 version of the architecture using 1023 Navigation Module B's. The CCA7 version using 1023 Navigation Module B's produced statistically significant ($p < .001$) shorter paths with a mean of 7432.2 miles (standard deviation 141.8 miles) than the CCA7 version using only one Navigation Module B producing paths with a mean of 8131.0 miles (standard deviation 0 miles since the nearest neighbor algorithm is followed the same each run). If a CCA7 version using one Navigation Module B includes random fluctuations at each decision point, then the results are even worse with a shortest path mean of 9965.5 miles (standard deviation 1535.5 miles). Thus, the data supports the premise the CCA7 version of the architecture with multiple Navigation Module B's, reflecting a hypothetical evolution of the previous architecture (intended to model the human brain, albeit very loosely in a functionalist fashion as per Lieto, 2021b), performs better in this traveling salesperson problem, and potentially, all planning problems.

Many humans who look at the traveling salesperson problem distance values in Table B1 can be shown how to perform the nearest neighbor algorithm, although without paper and a writing instrument once the number of cities exceeds the limits of working memory often have trouble recalling which cities have already been navigated to, in addition to requiring good attention to find the smallest next number. As well, even with good attention and good recall of which cities have been navigated to, humans cannot readily produce much better results from the numbers than such a nearest-neighbor solution. However, as MacGregor and colleagues (2004) note, humans can use a variety of cognitive processes to do better than expected on visual, Euclidean versions of the traveling salesperson problem. Nonetheless, as planning problems get larger or involve more dimensions, it becomes even harder for humans to obtain better results than is given by the CCA7 architecture.

Given that 1023 Navigation Module B's produce better results than a single Navigation Module B, it is interesting to consider the results of Table B4 and Figure 15. The mean shortest distance for the traveling salesperson problem in Table B1 (Appendix B, Supplementary Material) is 7432.2 miles (standard deviation 141.8 miles) for 1K Navigation Module B's versus 7309.1 miles (standard deviation 29.8 miles) for 4K Navigation Module B's versus 7296.8 miles (standard deviation 6.9 miles) for 16K Navigation Module B's. Note that the theoretical shortest path of 7293 miles is quite close to the mean results of the 4K and in particular the 16K architectures. Indeed, two-thirds of the runs done of the 16K version of the architecture yielded the theoretical shortest path. Thus, larger numbers of Navigation Module B's allow better planning solutions as represented in this particular traveling salesperson problem. In larger planning problems, the 16K architecture will not as easily be able to find the optimal solution, but nonetheless, having additional Navigation Module B's provides a larger solution space.

To avoid artificially creating arithmetic capabilities which the architecture did not possess, the utilization of the position weighting of the possible locations to navigate to next was discussed above. The results above of table B5 and Figure 15 compare a CCA7 version of the architecture using the position weighting method (given in equations (124) to (127)) versus value weighting method given in Section 4.2. In the value weighting method, the actual values of the distance to the next possible locations to navigate to are considered with each other. Table B5 shows that the mean of position weighting method gave a somewhat mean shortest distance of 7432.2 miles (standard deviation 141.8

miles) versus 7534 miles (standard deviation 58.0 miles) for the value weighting method. This difference was statistically significant at $p < 0.001$. Thus, the position weighting method used by the CCA7 is reasonable compared to a simple value weighting method.

6.2 Parallel Operation of Different Instinctive and Learned Primitives and their Simulation

As noted above, equation (116) indicates that the same instinctive primitive or the same learned primitive is initially applied to all of the Navigation Module B's (Figure 9).

$$(\mathbf{WNMB}'_{n=x} \neq \text{reserved} \text{ AND } \mathbf{WNMB}'_{n=y} \neq \text{reserved}) \\ \Rightarrow \text{initial_primitive}_{\mathbf{WNMB}'_{n=x}} = \text{initial_primitive}_{\mathbf{WNMB}'_{n=y}} \quad (116)$$

As discussed above, random fluctuations can be introduced in the different Navigation Module B's to produce a variety of results to choose from. In the example shown above and simulated, i.e., the traveling salesperson problem, despite these random fluctuations, the same instinctive primitives were used by all the Navigation Module B's, so it was not an issue. (The instinctive primitive `small_plan()` did use a different argument in the $n=1$ Navigation Module B but it is automatically selected by the Navigation Module per equation (118).)

However, what if a different type of problem is being operated on by the Navigation Module B's such that there are also random fluctuations in the different modules, and as a result, at certain logical decision points, different instinctive or learned primitives are triggered and must be obtained by the module? What if there are a million Navigation Module B's, for example, all attempting to trigger and retrieve navigation maps at the same time from the instinctive primitives and learned primitives from the Causal Memory Module, Instinctive Primitives Module, and the Learned Primitives Module?

At present, the Python simulation of the CCA7 version of the architecture simply runs everything sequentially. Training of neural networks used by various modules is performed ahead of time, although in the present version of the simulation, a pattern recognition library is used rather the previous PyTorch machine learning Python library. The speed of the of modern computers is fast enough to handle simple simulations of the architecture, even with a slow language such as Python (albeit the data structure of the code is based on the Python library Numpy which is written in efficient C code), so there has not been a need to pay attention to executing the simulation on parallel processors. In the present simulation, when all operations of a cognitive cycle have been completed, then the next cognitive cycle starts, without consideration of how long the cognitive cycle actually took. Thus, even if a different type of problem where the Navigation Module B's must retrieve and write different navigation maps and primitives from different modules of the architecture, is simulated, the simulation will sequentially perform the operations, and then consider the cognitive cycle completed. At present, there generally are not multiple comparisons between different modules within one cognitive cycle which could cause problems with a sequential simulation.

There is the need in future work on the architecture, to better specify real-time parallel operation of the different Navigation Module B's as well as to better simulate the architecture on parallel central processing units (CPUs)/graphics processing units (GPUs). A full consideration of parallel computing is beyond the scope of this article. However, the initial instinctive or learned primitive applied to all the Navigation Module B modules (i.e., $n=1 \dots 1023$ of Figure 9) is effectively a single

instruction stream, multiple data streams (SIMD) arrangement per Flynn’s classification (Flynn, 1972). However, as processing occurs this can essentially become a multiple instruction streams, multiple data streams (MIMD) arrangement. In future work, the CCA7 version of the architecture (i.e., Figure 9) and the associated equations/pseudocode describing it (i.e. Appendix A, Supplementary Material section) need to be enhanced to better meet MIMD requirements. In particular, the multiple **TempMap** areas in each of the Navigation Module B’s are designed to hold not a “byte” of information but rather a “navigation map” of information, i.e., they can hold copies of instinctive primitives, learned primitives as well as portions of the Causal Memory Module. This resource can obviate many of the difficulties in implementing MIMD requirements.

6.3 Improved Autonomic Module Control of the Navigation Module B’s and its Simulation

The current Python simulation of the CCA7 version of the architecture does not simulate the Autonomic Module (Figure 9) in much detail. Given its brain inspired origins, the Causal Cognitive Architecture has long had an Autonomic Module as does the CCA7 (Figure 9) (Schneider, 2021). Just as the autonomic nervous system in mammals plays a key role in maintaining homeostasis, so does the Autonomic Module in the Causal Cognitive Architecture involve itself with maintaining energy usage, heat production, sleep cycles and the reliability of the architecture’s physical embodiment.

Although sleep cycles do exist in the Python simulation of the Autonomic Module of the architecture, they are not very sophisticated. Either the architecture is on, or it is in a low energy sleep-like state. Indeed, even though biological mental activity is associated with region-specific increased energy consumption, the overall energy expenditure of the mammalian brain tends to be more constant than would be expected, whether the brain is problem-solving or idle but awake (Raichle, Gusnard, 2002). Indeed, when the mind is idle, a significant default mode network (DMN) of mammalian brain areas becomes active (Raichle, 2015). However, during sleep, the overall energy expenditure of the brain does indeed decrease (Dworak et al., 2010).

Thus, at present, the Autonomic Module in the Causal Cognitive Architecture, including the CCA7 version (Figure 9), simply has an awake/sleep functionality with regard to energy expenditure of the architecture. However, in the CCA7 version of the architecture, there are now over a thousand Navigation Module B’s, and there could be, for example, in another implementation, millions of such Navigation Module B’s all working in parallel. The large numbers of Navigation Module B’s are not based on current mammalian including human brains, but instead on the hypothetical possibility that might arise in response to the question where in the Introduction section the question is asked what if the evolution of the human brain were to continue as it has in the past, and given an environment for such evolution, as reflected in a model such as the Causal Cognitive Architecture.

A thousand, or especially a million, Navigation Module B’s operating all the time, would seem to be wasteful with regards to the consumption of energy, as well as perhaps cause problems with regard to the dissipation of heat produced. In future work, there is a need for the Autonomic Module of Figure 9 to directly connect with the Navigation Module B’s, so that it could regulate the activity of this ever-increasing portion of the architecture. Similarly, there is a need for principles of such regulation to be included in the equations and pseudocode of Appendix A (Supplementary Section). It is hypothesized that many planning situations and problems could be resolved with a limited number of Navigation Module B’s, so that perhaps the entire repertoire of modules could be selectively activated by the Autonomic Module in response to repeated feedback loops of inability to yield a

solution to a problem (e.g., equation (95) above), or directly by triggering by an instinctive or learned primitive.

6.4 Enhanced Intelligence from a Brain-Inspired Cognitive Architecture

As discussed above, the Causal Cognitive Architecture developed from the hypothesis that hundreds of millions of years ago the navigation circuits in the amniotic ancestors of mammals duplicated many times to eventually form the neocortex. Thus, millions of neocortical minicolumns are functionally modeled in the architecture as millions of spatial maps, i.e., the “navigation maps” of the architecture. From this starting point, the properties of a cognitive architecture based on these navigation maps and inspired by the mammalian brain, were investigated.

Without special feedback operations it was shown that a cognitive architecture based on navigation maps readily showed the reflexive and pre-causal behavior seen in most mammals (Schneider 2021). Then modest changes, inspired by modest genetic changes from the last chimpanzee-human common ancestor in the emergence of human cognitive abilities, were considered. Relatively modest changes were made which simply allow enhanced feedback operations and the addition of extra instinctive primitives. With these modest changes, full causal decision making emerged from the architecture (Schneider 2022a). Further exploration revealed very small changes which allowed the emergence of full inductive analogical reasoning abilities (Schneider, 2023) and the very ready emergence of compositional comprehension and language in the CCA6 version of the architecture (Schneider, 2024).

While the CCA6 version of the architecture is very conceptual other than for a Python simulation, it seems to hold many of the features unique to human cognition—full causal reasoning, full analogical reasoning, near-full compositional (as opposed to combinatorial) language, and unfortunately the development of a vulnerability to psychosis (Schneider, 2020). As well, note that navigation maps and consequential reasoning are fully grounded in the architecture (Schneider, 2023). In addition, note that the CCA6 is based on a cognitive architecture, which in its own right, forms the basis for an autonomous agent, i.e., tries to achieve goals by interactions with the environment (Paisner et al., 2014; Lieto et al., 2018).

In this paper, it was considered if further biologically plausible changes, as modest as possible, would allow significant improvements to the CCA6 architecture, such that super-human aspects of cognition would emerge. This paper is particularly interested in enhancing core aspects of cognition in a human brain-inspired cognitive architecture. Integrating a calculator, for example, or perhaps a state-of-the-art large language model (LLM) would certainly boost the cognitive abilities of the architecture in a certain sense. Indeed, hybridization of the CCA6 architecture with other tools would seem to yield promising results. Liu and colleagues (2023) note that cognitive architectures and generative models have complementary strengths and weaknesses, and discuss their fusion. These are valid topics for future consideration of the enhancement of the Causal Cognitive Architecture. However, this paper asks how modest, biologically plausible changes, could allow core aspects of cognition to surpass normal human abilities.

Genetic and developmental mechanisms have been put forward for the duplication of mammalian brain circuits and their divergence to new functions (e.g., Rakic, 2009, e.g., Chakraborty and Jarvis, 2015). Thus, an increase in temporary memories as well as the increase in Navigation Module B’s as a biologically plausible theoretical modification of the architecture has been hypothesized above,

resulting in the Causal Cognitive Architecture 7 (CCA7) (Figure 9). Evolution of new instinctive primitives such as “small_plan()” may be more involved from a mechanistic point of view. While learned primitives can more readily acquire millions or billions of bytes of information, genetic acquisition of an instinctive primitive has obvious resource and chance limitations (e.g., Weber and colleagues (2013) regarding the instinct for the burrowing habits of *Peromyscus* mice). However, chimpanzees plan tool utilization (e.g., for use at termite nests) so some instinctual planning abilities may have been present for a long time, i.e., if present in the last chimpanzee-human common ancestor there is a longer period for such instincts to have evolved over time (Musgrave et al., 2023). Thus, the instinctive primitive “small_plan()” really should have been included in earlier versions of the Causal Cognitive Architecture, and would have had enough time for any small changes to the version used in the CCA7.

As shown above, a large-scale duplication of the Navigation Module B circuits allows the architecture to have better planning abilities. This is readily apparent in Figure 14. The cognitive ability to imagine and plan for future events has long been considered a very advantageous aspect of cognition (e.g., Suddendorf and Corballis, 2007). Having super-human planning abilities, for example, such as the ability to simultaneously consider 1023 (or 16,383) navigation routes (or analogous routes for other actions) as shown above, can allow the architecture to analyze and plan for its environment at a higher level than normally possible for the CCA6 with its single Navigation Module B, or a human.

Future work on the Causal Cognitive Architecture 7’s simulation includes enlarging its set of instinctive primitives as well as better educational experiences via (and better functioning of) the learned primitives system. As well, with regard to planning there are a myriad of other algorithms possible to consider rather than just introducing random fluctuations via the *weight* parameter. For example, the nearest-neighbor algorithm and even its modification with random fluctuations, is susceptible to falling into a local optimum trap. For some planning problems, at the decision points, in addition to local information concerning which location is the closest, second closest, and so on, global information can be very advantageous to a more efficient shortest-path. For example, in the ant colony optimization solution mentioned briefly above, accumulated knowledge from previous attempts to solve a route problem is used in addition to the local information at decision points. Further work is needed in applying these strategies to the CCA7’s instinctive and/or learned primitives. As well, the effect of random fluctuations and planning in other areas besides the traveling salesperson problem needs to be considered. Once more of this work is done, it becomes more relevant to benchmark the architecture against other intelligent agents.

At present the CCA7 is largely conceptual, its Python simulation notwithstanding, and cannot do useful work compared to a modern LLM-based chatbot, for example. However, as noted above, it seems to hold many of the features unique to human cognition—full causal reasoning, full analogical reasoning, near-full compositional language, and now planning. As well it is fully grounded and autonomous. Thus, in a conceptual sense anyway, the CCA7 represents human level artificial intelligence (HLAI) abilities. Given that the simultaneous multi-planning abilities of the CCA7 can be used for many cognitive processes at a level exceeding what humans are capable of, the CCA7 shows in a conceptual sense, some sparks of superintelligence. (The emphasis should be on “sparks” rather than “superintelligence,” with homage to Bostrom (2014) and Bubeck and colleagues (2023).) As brain-inspired cognitive architectures such as the CCA7 become developed enough to realize their theoretical potential (or possibly fail at it) they should be considered as a viable alternative pathway towards the development of HLAI and then superintelligence, as well as giving insight into the emergence of natural human level intelligence.

7 Conflict of Interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

8 Author Contributions

H.S. originated and developed the ideas in this paper. H.S. wrote this paper.

9 Funding

No external funding.

10 References

- Adams, S., Arel, I., Bach, J., Coop, R., Furlan, R., Goertzel, B., Hall, J., Samsonovich, A., Scheutz, M., Schlesinger, M., Shapiro, S., Sowa, J. : Mapping the Landscape of Human-Level Artificial General Intelligence. *AI Magazine* (2012). 33:25-42. 10.1609/aimag.v33i1.2322.
- Alme, C. B., Miao, C., Jezek, K., Treves, A., Moser, E. I., & Moser, M. B. (2014). Place cells in the hippocampus: eleven maps for eleven rooms. *Proceedings of the National Academy of Sciences of the United States of America*, 111(52), 18428–18435. <https://doi.org/10.1073/pnas.1421056111>
- Ando, J., Ono, Y., & Wright, M. J. (2001). Genetic structure of spatial and verbal working memory. *Behavior genetics*, **31**, 615-624.
- Aubry, A., Gonthier, C., & Bourdin, B. (2021). Explaining the high working memory capacity of gifted children: Contributions of processing skills and executive control. *Acta psychologica*, **218**, 103358.
- Bang-Jensen, J., Gutin, G., & Yeo, A. (2004). When the greedy algorithm fails. *Discrete optimization*, *1*(2), 121-127.
- Behrens, T. E., Muller, T. H., Whittington, J. C., Mark, S., Baram, A. B., Stachenfeld, K. L., Kurth-Nelson, Z. (2018). What is a cognitive map? Organizing knowledge for flexible behavior. *Neuron*, **100**(2), 490-509.
- Bostrom, N. (2014). *Superintelligence: Paths, strategies, dangers*. Oxford University Press.
- Bubeck, S., Chandrasekaran, V., Eldan, R., Gehrke, J., Horvitz, E., Kamar, E., Lee, P., Lee, Y.T., Li, Y., Lundberg, S., Nori, H. (2023). Sparks of artificial general intelligence: Early experiments with GPT-4. *arXiv:2303.12712*

- Burmeister, S. S. (2022). Ecology, cognition, and the hippocampus: A tale of two frogs. *Brain Behavior and Evolution*, 97(3-4), 211-224.
- Butler AB, Reiner A, Karten HJ. (2011). Evolution of the amniote pallium and the origins of mammalian neocortex. *Ann N Y Acad Sci.* Apr;**1225**:14-27. doi: 10.1111/j.1749-6632.2011.06006.x
- Buzsaki, G., Moser, E.I. Memory, navigation and theta rhythm in the hippocampal-entorhinal system. *Nature Neuroscience*, **16**(2):130–138, 2013
- Carruthers, P. (2013). Evolution of working memory. *Proceedings of the National Academy of Sciences*, **110**(supplement_2), 10371-10378.
- Chakraborty, M., Jarvis, E.D. (2015). Brain evolution by brain pathway duplication. *Philosophical transactions of the Royal Society of London. Series B, Biological sciences*, **370**(1684), 20150056 <https://doi.org/10.1098/rstb.2015.0056>
- Chuderski, A., & Jastrzębski, J. (2018). Much ado about aha!: Insight problem solving is strongly related to working memory capacity and reasoning ability. *Journal of Experimental Psychology: General*, **147**(2), 257.
- Claxton, L. J., Keen, R., & McCarty, M. E. (2003). Evidence of motor planning in infant reaching behavior. *Psychological Science*, **14**(4), 354-356.
- Cowan, N. (2001). The magical number 4 in short-term memory: A reconsideration of mental storage capacity. *Behavioral and brain sciences*, **24**(1), 87-114.
- Dhein, K. (2023). The cognitive map debate in insects: A historical perspective on what is at stake. *Studies in History and Philosophy of Science*, **98**, 62-79.
- Dorigo, M., & Gambardella, L. M. (1997). Ant colonies for the travelling salesman problem. *biosystems*, **43**(2), 73-81.
- Dry, M., Lee, M. D., Vickers, D., & Hughes, P. (2006). Human performance on visually presented traveling salesperson problems with varying numbers of nodes. *The Journal of Problem Solving*, **1**(1), 4.
- Dworak, M., McCarley, R. W., Kim, T., Kalinchuk, A. V., Basheer, R. (2010). Sleep and brain energy levels: ATP changes during sleep. *Journal of Neuroscience*, **30**(26), 9007-9016.
- Flemming, T.M., Thompson, R.K., Fagot, J. (2013). Baboons, like humans, solve analogy by categorical abstraction of relations. *Anim Cogn.* **16**(3):519-24. doi: 10.1007/s10071-013-0596-0.
- Flynn, M. J. (1972). Some computer organizations and their effectiveness. *IEEE transactions on computers*, **100**(9), 948-960.
- Fournier, J., Müller, C. M., & Laurent, G. (2015). Looking for the roots of cortical sensory computation in three-layered cortices. *Current opinion in neurobiology*, **31**, 119-126.
- Frégnac, Y. (2023). Flagship Afterthoughts: Could the Human Brain Project (HBP) Have Done Better? *Eneuro*, **10**(11).

- Friedman, N. P., & Miyake, A. (2017). Unity and diversity of executive functions: Individual differences as a window on cognitive structure. *Cortex*, **86**, 186-204.
- Friedman, N. P., Miyake, A., Young, S. E., DeFries, J. C., Corley, R. P., & Hewitt, J. K. (2008). Individual differences in executive functions are almost entirely genetic in origin. *Journal of experimental psychology: General*, **137**(2), 201.
- Friston, K. (2010). The free-energy principle: a unified brain theory? *Nature reviews neuroscience*, **11**(2), 127–138.
- Georgeon, O. L., Lurie, D., and Robertson, P. (2024). Artificial enactive inference in three-dimensional world. *Cognitive Systems Research*, 101234.
<https://doi.org/10.1016/j.cogsys.2024.101234>
- Google OR-Tools (2023). Traveling Salesperson Problem. Retrieved Dec 20, 2023 from: <https://developers.google.com/optimization/routing/tsp>
- Güntürkün, O., von Eugen, K., Packheiser, J., & Pusch, R. (2021). Avian pallial circuits and cognition: a comparison to mammals. *Current Opinion in Neurobiology*, **71**, 29-36.
- Hagmann, C.E, Cook, R.G. (2015). Endpoint distinctiveness facilitates analogical mapping in pigeons. *Behav Processes*. **112**:72-80. doi: 10.1016/j.beproc.2014.11.007.
- Harris, C. R., Millman, K. J., Van Der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., ... Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, **585**(7825), 357-362.
- Hawkins, J., Lewis, M., Klukas, M., Purdy, S., Ahmad, S. (2019). A framework for intelligence and cortical function based on grid cells in the neocortex. *Frontiers in neural circuits*, **12**, 121.
- Hofstadter, D.R. (2001). Analogy as the core of cognition. In Gentner, D., Holyoak, K.J., and Kokinov, B.N. editors, *The Analogical Mind: Perspectives from Cognitive Science*, pp 499–538. MIT Press.
- Joshi, H., Ustun, V. (2023). Augmenting Cognitive Architectures with Large Language Models. In *Proceedings of the AAAI Symposium Series*, Vol. 2, No. 1, pp. 281-285.
- Kaas, J. H. (2019). The origin and evolution of neocortex: From early mammals to modern humans. *Progress in brain research*, **250**, 61-81.
- Kahneman, D. (2011). *Thinking, fast and slow*. Farrar, Straus and Giroux, NY, USA.
- Kinzler, K.D., Spelke, E.S., (2007). Core systems in human cognition. In von Hofsten, C., Rosander, K. *Progress in Brain Research*, vol **164**, chap 14.
- Kotseruba, I., Tsotsos, J.K. (2020). 40 years of cognitive architectures: core cognitive abilities and practical applications. *Artif Intell Rev* **53**, 17-94. <https://doi.org/10.1007/s10462-018-9646-y>
- Kwon, K. (2014). Expressing Algorithms as Concise as Possible via Computability Logic. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, **97**(6), 1385-1387. Open source: *ArXiv*: 1305.2004

- Laird, J. E., Lebiere, C., Rosenbloom, P. S. (2017) A Standard Model of the Mind: Toward a Common Computational Framework across Artificial Intelligence, Cognitive Science, Neuroscience, and Robotics. *AI Magazine*, **38**(4), 13-26. <https://doi.org/10.1609/aimag.v38i4.2744>
- Laird, J. E., Wray, R. E., Jones, S., Kirk, J. R., Lindes, P. (2023). Proposal for Cognitive Architecture and Transformer Integration: Online Learning from Agent Experience. In *Proceedings of the AAAI Symposium Series*, Vol. 2, No. 1, pp. 302-306.
- Legg, S., Hutter, M.: Universal Intelligence: A Definition of Machine Intelligence (2007). arXiv: 0712.3329
- Lieto, A., Bhatt, M., Oltramari, A., Vernon, D. (2018). The role of cognitive architectures in general artificial intelligence. *Cognitive Systems Research*, **48**, 1-3.
- Lieto, A. (2021). *Cognitive design for artificial minds*. Routledge.
- Lieto, A. (2021b). Functional and Structural Models of Commonsense Reasoning in Cognitive Architectures. In Laird, J. (Ed.) *Virtual International Symposium on Cognitive Architecture VISCA 2021*. Retrieved Aug 28, 2023: <https://visca.engin.umich.edu/wp-content/uploads/sites/27/2021/06/Lieto.pdf>
- Liu, S., Pepe, B., Kumar, M.G., Ullman, T.D., Tenenbaum, J.B., Spelke, E.S. (2022). Dangerous Ground: One-Year-Old Infants are Sensitive to Peril in Other Agents' Action Plans. *Open Mind*, **6**:211-231
- Liu, Y., Liu, Y., Shen, C. (2023). Combining Minds and Machines: Investigating the Fusion of Cognitive Architectures and Generative Models for General Embodied Intelligence. In *Proceedings of the AAAI Symposium Series*, Vol. 2, No. 1, pp. 307-314.
- Ma W.J., Husain, M., Bays, P.M. (2014). Changing concepts of working memory. *Nature Neuroscience*. **17** (3): 347–356. doi:10.1038/nn.3655.
- MacGregor, J.N., Chronicle, E.P., Ormerod, T.C. (2004). Convex hull or crossing avoidance? Solution heuristics in the traveling salesperson problem. *Memory & Cognition* **32**, 260–270 doi.org/10.3758/BF03196857
- Madl, T., Baars, B.J., Franklin, S. (2011). The Timing of the Cognitive Cycle. *PLoS ONE*, **6**(4): e14803. doi: 10.1371/journal.pone.0014803
- Marcus, G., Davis, E., Aaronson, S. (2022). A very preliminary analysis of Dall-e 2. *arXiv*:2204.13807
- Markram, H. (2012). The human brain project. *Scientific American*, **306**(6), 50-55.
- McCormack, T., & Atance, C. M. (2011). Planning in young children: A review and synthesis. *Developmental Review*, **31**(1), 1-31.
- Millidge, B., Seth, A., & Buckley, C. L. (2021). Predictive coding: a theoretical and experimental review. arXiv preprint arXiv:2107.12979.
- Musgrave, S., Koni, D., Morgan, D., & Sanz, C. (2023). Planning abilities of wild chimpanzees (*Pan troglodytes troglodytes*) in tool-using contexts. *Primates*, 1-15.

- Oberauer, K., Farrell, S., Jarrold, C., & Lewandowsky, S. (2016). What limits working memory capacity? *Psychological bulletin*, **142**(7), 758.
- O'Keefe, J., Nadel, L. (1978). *The Hippocampus as a Cognitive Map*. Oxford Univ Press, Oxford.
- Olsen, A.L. (2005). Using pseudocode to teach problem solving. *Journal of Computing Sciences in Colleges*, *21*, 231-236.
- Paisner, M., Cox, M., Maynard, M., Perlis, D. (2014). Goal-driven autonomy for cognitive systems. In *Proceedings of the Annual Meeting of the Cognitive Science Society*, Vol. 36, No. 36.
- Penn, D.C., Holyoak, K.J., Povinelli, D.J. (2008). Darwin's mistake: explaining the discontinuity between human and nonhuman minds. *Behav Brain Sci.* **31**(2):109-30; discussion 130-178. doi: 10.1017/S0140525X08003543
- Raichle, M. E. (2015). The brain's default mode network. *Annual review of neuroscience*, *38*, 433-447.
- Raichle, M. E., & Gusnard, D. A. (2002). Appraising the brain's energy budget. *Proceedings of the National Academy of Sciences*, *99*(16), 10237-10239.
- Rakic, P. (2009). Evolution of the neocortex: a perspective from developmental biology. *Nat Rev Neurosci.* Oct;**10**(10):724-35. doi: 10.1038/nrn2719
- Rao, R. P., Ballard, D. H. (1999). Predictive coding in the visual cortex: a functional interpretation of some extraclassical receptive-field effects. *Nature neuroscience*, **2**(1), 79–87.
- Rosenkrantz, D. J., Stearns, R. E., & Lewis, II, P. M. (1977). An analysis of several heuristics for the traveling salesman problem. *SIAM journal on computing*, **6**(3), 563-581.
- Samsonovich, A.V. (2010). Toward a Unified Catalog of Implemented Cognitive Architectures. In *Proceedings of the 2010 Conference on Biologically Inspired Cognitive Architectures 2010: Proceedings of the First Annual Meeting of the BICA Society*. IOS Press, NLD, 195-244.
- Schafer, M., Schiller, D. (2018). Navigating Social Space. *Neuron*, **100**(2):476-489
- Schneider, H. (2020). The Meaningful-Based Cognitive Architecture Model of Schizophrenia. *Cognitive Systems Research* **59** 73-90 doi: 10.1016/j.cogsys.2019.09.01
- Schneider, H. (2021). Causal Cognitive Architecture 1: Integration of connectionist elements into a navigation-based framework. *Cognitive Systems Research* **66**:67-81 doi: 10.1016/j.cogsys.2020.10.021
- Schneider, H. (2022a). Causal cognitive architecture 3: A Solution to the binding problem. *Cognitive Systems Research* **72**:88-115 doi: 10.1016/j.cogsys.2021.10.004
- Schneider, H. (2022b). Navigation Map-Based Artificial Intelligence. *AI*, **3**(2) 434-464 doi:10.3390/ai3020026

- Schneider, H. (2023). An Inductive Analogical Solution to the Grounding Problem. *Cognitive Systems Research*, **77**:74-216 doi: 10.1016/j.cogsys.2022.10.005
- Schneider, H. (2024). The emergence of compositionality in a brain-inspired cognitive architecture. *Cognitive Systems Research*, **86**, 101215. <https://doi.org/10.1016/j.cogsys.2024.101215>
- Schuck, N. W., Cai, M. B., Wilson, R. C., Niv, Y. (2016). Human orbitofrontal cortex represents a cognitive map of state space. *Neuron*, **91**(6), 1402-1412.
- Spelke, E.S. (1994). Initial knowledge. *Cognition*, **50**, 431-45. doi:10.1016/0010-0277(4)90039-6
- Spelke, E. S., Kinzler, K. D. (2007). Core knowledge. *Developmental science*, 10(1), 89-96.
- Suddendorf, T., Corballis, M. C. (2007). The evolution of foresight: What is mental time travel, and is it unique to humans? *Behavioral and brain sciences*, 30(3), 299-313.
- Sun, R. (2024). Can A Cognitive Architecture Fundamentally Enhance LLMs? Or Vice Versa? *arXiv preprint arXiv:2401.10444*
- Tolman, E. C. (1948). Cognitive maps in rats and men. *Psychological review*, **55**(4), 189.
- Tschoke, S., Lubling, R., & Monien, B. (1995). Solving the traveling salesman problem with a distributed branch-and-bound algorithm on a 1024 processor network. In *Proceedings of 9th International Parallel Processing Symposium* (pp. 182-189). IEEE.
- Wang, P.: On Defining Artificial Intelligence. *Journal of Artificial General Intelligence*. 10(2):1-37 (2019). doi: 10.2478/jagi-2019-0002
- Weber, J., Peterson, B., Hoekstra, H. (2013). Discrete genetic modules are responsible for complex burrow evolution in *Peromyscus* mice. *Nature* **493**,402–405 doi: 10.1038/nature 11816
- Whittington, J. C., McCaffary, D., Bakermans, J. J., Behrens, T. E. (2022). How to build a cognitive map. *Nature neuroscience*, **25**(10), 1257-1272.

Modeling Equations including Pseudocode Overview

To provide a more formal yet understandable and relevant description of the Causal Cognitive Architecture 7 (CCA7), the quasi-formalization is provided below. This formalization provides the basis for the Python simulations of the Causal Cognitive Architecture 7. **Transition to CCA8 – to merge versions --**

The CCA7 architecture (Figure 9) is described by the modified equations below. They are “modified” in the sense that many of the equations contain pseudocode. A pseudocode is a common language (e.g., English) description of the logic of a software routine (Kwon, 2014; Olsen, 2005). Note that all symbols and all pseudocode of all equations are explained in the text and in the associated tables. The use of a more abstracted pseudocode in the equations below largely provides a more understandable description of the architecture without sacrificing much accuracy.

Many of the properties of the CCA7 architecture, and the corresponding equations below, are similar to the prior CCA6 architecture. However, the planning and parallel operations are new, and previous equations have been modified where required, and new equations were added.

The equations below represent one cognitive cycle through the CCA7 architecture, although some of the backward flow of the feedback information is also described. Not every path or module of the architecture is described. For example, the equations do not describe flow of sensory data through the Autonomous Reflex Modules (Figure 1). In actual operation many cognitive cycles of the architecture occur over and over again each second.

Bold capitalized letters represent arrays, e.g., in equation (18) below **LNM** represents an array which acts as a navigation map. Vectors are represented in bolded italics (e.g., vector *s* in equation (9)). Many of the values change with time and are shown with the subscript *t*. For example, $\mathbf{S}'_{\sigma,t}$ (18) represents an array \mathbf{S}'_{σ} with a value changing with time *t*. The subscript *t-1* represents the previous cognitive cycle and the subscript *t+1* represents the next cognitive cycle.

A.1 Input Sensory Vectors Shaping Modules

This section of Appendix A applies to the CCA5 (shown in Figure 1), the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

Figure 1 shows sensory inputs streaming into the Sensory Vectors Shaping Modules of the architecture. Sensory inputs for any sense are propagated into the architecture as a two-dimensional or three-dimensional spatial array of inputs. Spatial information means some sort of sensory information about a small volume of space which can be addressed and modeled as cell x,y,z. For example, for the visual sensory system, this spatial information would be visual sensory inputs (e.g., lines or no lines as a very simple example) observed at particular locations in the environment.

The details of signal processing, from the quantum level to the output produced by a transducer after possibly multi-layered signal processing, are largely abstracted away in this formalization. This does

not diminish the importance of better signal processing. However, the architecture is concerned with whatever processed sensory inputs stream in, and that is what is considered here.

An array \mathbf{S}_σ receives the sensory inputs of a sensory system σ every cognitive cycle, i.e., essentially changing with respect to time t (1 – 9). Vector $\mathbf{s}(t)$ is processed by the Input Sensory Vectors Shaping Modules (via pseudocode `Input_Sens_Shaping_Mods.normalize()`, described below in Table AA1) into vector $\mathbf{s}'(t)$ (10). This transformation ensures that each element of \mathbf{s}' will be compatible with the dimensions used by the navigation maps in many of the modules of the CCA6 (11).

As Table A2 shows, processed and normalized sensory system arrays $\mathbf{S}'_{\sigma,t}$ leave this module compatible with the other data structures, i.e., the navigation maps, of the architecture. $\mathbf{S}'_{\sigma,t}$ are in arrays of dimensions $m \times n \times o \times p$ corresponding to the three spatial dimensions x, y, z and a fourth non-spatial dimension p (used in implementations of the architecture for the storage of segmentation data, i.e., defining objects in a scene, and metadata).

These equations remain largely unchanged in the CCA7 architecture from the prior CCA5 and CCA6 architectures.

$$\mathbf{S}_1 \in \mathbb{R}^{m1 \times n1 \times o1} \quad (1)$$

$$\mathbf{S}_{1,t} = \text{visual inputs}(t) \quad (2)$$

$$\mathbf{S}_2 \in \mathbb{R}^{m2 \times n2 \times o2} \quad (3)$$

$$\mathbf{S}_{2,t} = \text{auditory inputs}(t) \quad (4)$$

$$\mathbf{S}_3 \in \mathbb{R}^{m3 \times n3 \times o3} \quad (5)$$

$$\mathbf{S}_{3,t} = \text{olfactory inputs}(t) \quad (6)$$

$$\sigma = \text{sensory system identification code} \in \mathbb{N} \quad (7)$$

$$\Theta_\sigma = \text{total number of sensory systems} \in \mathbb{N} \quad (8)$$

$$\mathbf{s}(t) = [\mathbf{S}_{1,t}, \mathbf{S}_{2,t}, \mathbf{S}_{3,t}, \dots, \mathbf{S}_{\Theta_\sigma,t}] \quad (9)$$

$$\mathbf{s}'(t) = \text{Input_Sens_Shaping_Mods.normalize}(\mathbf{s}(t)) = [\mathbf{S}'_{1,t}, \mathbf{S}'_{2,t}, \mathbf{S}'_{3,t}, \dots, \mathbf{S}'_{\Theta_\sigma,t}] \quad (10)$$

$$\mathbf{S}'_{\sigma,t} \in \mathbb{R}^{m \times n \times o \times p} \quad (11)$$

$\mathbf{S}_1 \in \mathbb{R}^{m1 \times n1 \times o1}$	<p>-defining \mathbf{S}_1 to be an array of dimensions $m1 \times n1 \times o1$</p> <p>-the form of our data structures is important; as will be seen below, most of the architecture utilizes a ‘navigation map’ data structure, i.e., arrays, of size $m \times n \times o \times p$</p> <p>-as noted in the text above, an array \mathbf{S}_σ receives the sensory inputs of a sensory system σ every cognitive cycle; in this case array \mathbf{S}_1 receives the sensory inputs of sensory system 1, which are the visual sensory inputs</p>
--	--

	-as noted in the text above, the details of sensory perception, i.e., sensory signal processing, from the quantum level to the output produced by a transducer after possibly multi-layered signal processing, are largely abstracted away in this formalization.
$S_{1,t}$	an array holding visual sensory inputs, which change with time
$S_{2,t}$	an array holding auditory sensory inputs, which change with time
$S_{3,t}$	-an array holding olfactory sensory inputs, which change with time -for simplification implementing the olfactory sensory inputs similar to other sensory system, although this does not reflect actual mammalian neurophysiology
σ	=1 for visual, =2 for auditory, =3 for olfactory, other senses not used at present
Θ_σ	total sensory systems which above would be 3 for the current simulation
$s(t)$	a vector holding all the sensory input arrays
<code>Input_Sens_Shaping_Mods.normalize()</code>	normalizes the sensory input arrays to the same dimensions used by the other navigation maps in the architecture
$s'(t)$	a vector holding all the normalized sensory input arrays, each now with dimensions $m \times n \times o \times p$ (which in the simulation in the paper are implemented as $6 \times 6 \times 6$, corresponding to x,y,z axes of $6 \times 6 \times 6$, and another dimension used for object segmentation data $\times 16$)
$S'_{\sigma,t}$	normalized array (compatible with navigation maps in the other modules of the architecture) of sensory inputs of sensory system σ (which change with time)

Table A1. Explanation of Symbols and Pseudocode in Equations (1) – (11)

Input:	sensory inputs from sensory systems $1 \dots \Theta_\sigma$
Output:	$s'(t)$ -- a vector holding all the normalized sensory input arrays $S'_{1,t} \dots S'_{\Theta_\sigma,t}$
Description:	<p>-Raw sensory inputs from the environment are normalized into a format compatible with navigation map data structure used by the other modules of the architecture.</p> <p>-The current simulation of the architecture includes an experimental environment simulation module (i.e., simulates the sensory inputs).</p> <p>-Applies to CCA5, CCA6 and CCA7 versions of the architecture</p>

Table A2. Summary of the Operations of the Input Sensory Vectors Shaping Modules per Equations (1) – (11)

A.2 The Input Sensory Vectors Association Modules

This section of Appendix A applies to the CCA5 (shown in Figure 1), the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

Figure 1 shows normalized sensory inputs for each sensory system from the Input Sensory Vectors Shaping Modules to the Input Sensory Vectors Association Modules. The normalized sensory input arrays $\mathbf{S}'_{1,t} \dots \mathbf{S}'_{\Theta,t}$ are fed into corresponding modules (one module for each sensory system) of the Input Sensory Vectors Association Modules. Each module will map the sensory inputs it is receiving into a navigation map. This navigation map is called a “local navigation map” $\mathbf{LNM}_{(\sigma, mapno)}$ (i.e., local navigation map \mathbf{LNM} with address $mapno$ in sensory system σ) (14). Each cell in three spatial dimensions in $\mathbf{LNM}_{(\sigma, mapno)}$ can represent the full contents of each cell, i.e., all the features, procedures, and link addresses associated with a cell. As (14) shows there is also a non-spatial dimension p which is used in implementations to store various non-spatial information.

$\mathbf{all_maps}_{\sigma,t}$ is a vector holding all the local navigation maps in the σ sensory system Input Sensory Vectors Association Module (15). For example, $\mathbf{all_maps}_{1,t}$ represents all the stored local navigation maps in the visual Input Sensory Vectors Association Module (which as noted above is $\sigma = 1$).

As noted above $\mathbf{S}'_{1,t}$ is an array of the visual processed inputs (i.e., $\sigma = 1$). $\mathbf{S}'_{2,t}$ are the auditory processed inputs since $\sigma = 2$, and so on. The visual processed inputs $\mathbf{S}'_{1,t}$ should be matched against $\mathbf{all_maps}_{1,t}$, i.e., against all the other visual local navigation maps stored in the visual Input Sensory Vectors Association Module. The auditory, olfactory and any other sensory inputs should be matched against the respective local area maps stored in a particular Input Sensory Vectors Association Module.

In (18) $\text{Input_Assocn_Mod}_{\sigma}.\text{match_best_local_navmap}$ is pseudocode that matches an incoming sensory array σ (e.g., if $\sigma=1$ then $\mathbf{S}'_{1,t}$ is an array of the visual processed inputs) against the respective stored local navigation maps $\mathbf{all_maps}_{\sigma,t}$ (e.g., if $\sigma=1$ then $\mathbf{all_maps}_{1,t}$ holds all the stored local navigation maps in the visual Input Sensory Vectors Association Module). $\mathbf{LNM}_{(\sigma, \gamma, t)}$ represents the local navigation map \mathbf{LNM} in sensory system σ with a $mapno$ of γ which is the best match of $\mathbf{S}'_{\sigma,t}$. For example, if sensory inputs array $\mathbf{S}'_{1,t}$ best matches to $mapno$ 3456 (as an example) in the local navigation maps stored in the visual Input Sensory Vectors Association Module, then $\mathbf{LNM}_{(1, \gamma, t)}$ would be local navigation map 3456 in the visual Input Sensory Vectors Association Module.

Above it was noted that the Causal Cognitive Architecture makes heavy use of feedback pathways—states of a downstream module can influence the recognition and processing of more upstream sensory inputs. Thus, the previous cognitive cycle’s Working Navigation Map \mathbf{WNM}'_{t-1} (the navigation map in the Navigation Module A in Figure 9) and on which the Navigation Module A can perform operations on) is used in the pseudocode method (18) in deciding which is a best match.

These equations remain largely unchanged in the CCA6 and CCA7 architectures from the prior CCA5 architecture, although there is the allowance of multiple navigation modules in performing operations.

$mapno = \text{map identification code} \in \mathbb{N}$ (12)

$\Theta = \text{total number of used local navigation maps in a sensory system } \sigma \in \mathbb{N}$ (13)

$\mathbf{LNM}_{(\sigma, mapno)} \in \mathbb{R}^{m \times n \times o \times p}$ (14)

$all_maps_{\sigma, t} = [\mathbf{LNM}_{(\sigma, 1, t)}, \mathbf{LNM}_{(\sigma, 2, t)}, \mathbf{LNM}_{(\sigma, 3, t)}, \dots, \mathbf{LNM}_{(\sigma, \Theta, t)}]$ (15)

$\gamma = \text{map number of best matching map in a given set of navigation maps} \in mapno$ (16)

$\mathbf{WNM}' = \in \mathbb{R}^{m \times n \times o \times p}$ (17 and defined again below)

$\mathbf{LNM}_{(\sigma, \gamma, t)} = \text{Input_Assocn_Mod}_{\sigma}.\text{match_best_local_navmap}(\mathbf{S}'_{\sigma, t}, all_maps_{\sigma, t}, \mathbf{WNM}'_{t-1})$ (18)

$\mathbf{LNM}_{(\sigma, mapno)}$	a LNM or “local navigation map” which is a navigation map (i.e., an array of x,y,z dimensions m x n x o, which in the simulation in the paper are 6 x 6 x 6, corresponding to x,y,z axes of 6 x 6 x 6) that is held in the Input Sensory Vectors Association Module σ (e.g., if $\sigma = 1$, then that would be the module receiving the visual sensory inputs) and is map number $mapno$ (there may be millions of other local navigation maps stored in that Input Sensory Vectors Association Module σ)
$all_maps_{\sigma, t}$	a vector holding all the LNMs in module σ (i.e., starting with the first LNM and going to the last utilized \mathbf{LNM}_{Θ}) e.g., $all_maps_{1, t}$ would be all the LNMs in the Input Sensory Vectors Association Module 1 which are all the local navigation maps created and stored from the visual sensory inputs -there are new sensory inputs each cognitive cycle, and thus new local navigation maps will be produced, and thus there is a t subscript representing a change with time
\mathbf{WNM}'	a WNM is a “Working Navigation Map” it is a navigation map which for the moment is in the Navigation Module (Figure 1) and on which the Navigation Module can perform operations on As will defined below, operations can be performed by Navigation Module A or Navigation Module B, where are referred to as ‘Navigation Module’ in this section
$\text{Input_Assocn_Mod}_{\sigma}.\text{match_best_local_navmap}(\mathbf{S}'_{\sigma, t}, all_maps_{\sigma, t}, \mathbf{WNM}'_{t-1})$	pseudocode for an algorithm which will match the σ sensory inputs (e.g., if $\sigma = 1$ then it would be visual sensory inputs) against all the local navigation maps stored in the σ module of the Input Sensory Vectors Association Modules (Figure 1) and returns the best matched local navigation map \mathbf{LNM} for that sensory module -note that the results of the Navigation Module in the previous cycle as represented by previous cognitive cycle’s Working Navigation Map \mathbf{WNM}'_{t-1} are considered in the determining the best match
γ	represents a map number of a navigation map in an Input Sensory Vectors Association Module
$\mathbf{LNM}_{(\sigma, \gamma, t)}$	the local navigation map \mathbf{LNM} stored in the Input Sensory Vectors Association Module σ with map number γ which best matches in the incoming σ sensory inputs (e.g., if $\sigma = 1$ then it would be visual sensory inputs)

	e.g. if the visual sensory inputs best match the local navigation map #3456 in the visual module of the Input Sensory Vectors Association Modules, then at that moment $\mathbf{LNM}_{(1, \gamma, t)}$ would be navigation map #3456 in the visual module
--	---

Table A3. Explanation of Symbols and Pseudocode in Equations (12) – (18)

At this point, in every sensory module in the Input Sensory Vectors Association Modules (Figure 1), there is a best matching $\mathbf{LNM}_{(\sigma, \gamma, t)}$ (“local navigation map” since these navigation maps are stored locally in the sensory module rather than being stored in the Causal Memory Module attached to the Navigation Module, as seen in Figure 1). The next operation is to update the best matching local navigation map $\mathbf{LNM}_{(\sigma, \gamma, t)}$ with the actual sensory inputs $\mathbf{S}'_{\sigma, t}$ (21), creating an updated best matching navigation map $\mathbf{LNM}'_{(\sigma, \gamma, t)}$ which is renamed (for simulation compatibility issues) to $\mathbf{LNM}_{(\sigma, \gamma, t)}$ again.

If too many differences exist between the actual sensory inputs $\mathbf{S}'_{\sigma, t}$ and the best matching navigation map $\mathbf{LNM}_{(\sigma, \gamma, t)}$, then instead of updating the matched navigation map $\mathbf{LNM}_{(\sigma, \gamma, t)}$, a new local navigation map $\mathbf{LNM}_{(\sigma, new_map, t)}$ is created and updated with the actual sensory inputs $\mathbf{S}'_{\sigma, t}$ forming an updated best matching navigation map $\mathbf{LNM}'_{(\sigma, \gamma, t)}$ (22), which is renamed (for simulation compatibility issues) to $\mathbf{LNM}_{(\sigma, \gamma, t)}$ again.

In each sensory system Input Sensory Vectors Association Module the updated local navigation map (or newly created and updated one) is stored in the Input Sensory Vectors Association Module σ , and in future cognitive cycles, sensory inputs will be matched against it and the other local navigation maps stored there.

Vector \mathbf{lnm}_t represents the best-matching and updated local navigation maps $\mathbf{LNM}_{(\sigma, \gamma, t)}$ of all the different sensory modules of the Input Sensory Vectors Association Modules (23).

These equations remain largely unchanged in the CCA6 architecture from the prior CCA5 architecture, although there is the allowance of multiple navigation modules in performing operations.

$$h = \text{number of differences allowed to be copied onto existing map} \in \mathbb{R} \quad (19)$$

$$new_map = \text{map number of new local navigation map added to current sensory system } \sigma \in mapno \quad (20)$$

$$\begin{aligned} & | \text{Input_Assocn_Mod}_{\sigma} . \text{differences} (\mathbf{S}'_{\sigma, t}, \mathbf{LNM}_{(\sigma, \gamma, t)}) | \leq h, \\ \Rightarrow \mathbf{LNM}_{(\sigma, \gamma, t)} &= \mathbf{LNM}'_{(\sigma, \gamma, t)} = \mathbf{LNM}_{(\sigma, \gamma, t)} \cup \mathbf{S}'_{\sigma, t} \quad (21) \end{aligned}$$

$$\begin{aligned} & | \text{Input_Assocn_Mod}_{\sigma} . \text{differences} (\mathbf{S}'_{\sigma, t}, \mathbf{LNM}_{(\sigma, \gamma, t)}) | > h, \\ \Rightarrow \mathbf{LNM}_{(\sigma, \gamma, t)} &= \mathbf{LNM}'_{(\sigma, \gamma, t)} = \mathbf{LNM}_{(\sigma, new_map, t)} \cup \mathbf{S}'_{\sigma, t} \quad (22) \end{aligned}$$

$$\mathbf{lnm}_t = [\mathbf{LNM}_{(1, \gamma, t)}, \mathbf{LNM}_{(2, \gamma, t)}, \mathbf{LNM}_{(3, \gamma, t)}, \dots, \mathbf{LNM}_{(\theta-\sigma, \gamma, t)}] \quad (23)$$

h	the number of differences allowed to be copied onto existing map – if there are too many differences between the best matching local navigation map retrieved from the sensory system’s
---	---

	Input Sensory Vectors Association Module (Figure 1) and the actual input sensory signal (i.e., $\mathbf{S}'_{\sigma,t}$) then rather than copying the information from input sensory array onto the best matching local navigation map, and simply makes $\mathbf{S}'_{\sigma,t}$ into a new local navigation map
new_map	the <i>mapno</i> of an empty local navigation map in a sensory system's Input Sensory Vectors Association Module used when making $\mathbf{S}'_{\sigma,t}$ into a new local navigation map
$Input_Assocn_Mod_{\sigma} .differences$	-pseudocode for an algorithm that calculates the differences between two navigation maps -used in (21) and (22) to calculate the differences between the input sensory signal $\mathbf{S}'_{\sigma,t}$ and the retrieved best matching local navigation map $\mathbf{LNM}_{(\sigma, \gamma, t)}$
$\mathbf{LNM}_{(\sigma, \gamma, t)} \cup \mathbf{S}'_{\sigma,t}$ $\rightarrow \mathbf{LNM}' \rightarrow \mathbf{LNM}$	update $\mathbf{LNM}_{(\sigma, \gamma, t)}$ with any new information in input sensory signal $\mathbf{S}'_{\sigma,t}$ (i.e., copy $\mathbf{S}'_{\sigma,t}$ onto $\mathbf{LNM}_{(\sigma, \gamma, t)}$) thereby creating an updated $\mathbf{LNM}'_{(\sigma, \gamma, t)}$ (for simulation compatibility issues renaming $\mathbf{LNM}'_{(\sigma, \gamma, t)}$ to $\mathbf{LNM}_{(\sigma, \gamma, t)}$ again)
$\mathbf{LNM}_{(\sigma, new_map, t)} \cup \mathbf{S}'_{\sigma,t}$ $\rightarrow \mathbf{LNM}' \rightarrow \mathbf{LNM}$	copy $\mathbf{S}'_{\sigma,t}$ onto an empty $\mathbf{LNM}_{(\sigma, new_map, t)}$ thereby creating an updated $\mathbf{LNM}'_{(\sigma, \gamma, t)}$ (for simulation compatibility issues renaming $\mathbf{LNM}'_{(\sigma, \gamma, t)}$ to $\mathbf{LNM}_{(\sigma, \gamma, t)}$ again)
lnm_t	i.e., a vector holding $[\mathbf{LNM}_{(1, \gamma, t)}, \mathbf{LNM}_{(2, \gamma, t)}, \mathbf{LNM}_{(3, \gamma, t)}, \dots, \mathbf{LNM}_{(\Theta_{\sigma}, \gamma, t)}]$ -- the local navigation maps \mathbf{LNM} of each sensory system which best matches the corresponding sensory input array $\mathbf{S}'_{\sigma,t}$ and then are updated to \mathbf{LNM}' with the actual sensory information conveyed by $\mathbf{S}'_{\sigma,t}$ (for simulation compatibility issues renaming $\mathbf{LNM}'_{(\sigma, \gamma, t)}$ to $\mathbf{LNM}_{(\sigma, \gamma, t)}$ again)

Table A4. Explanation of Symbols and Pseudocode in Equations (19) – (23)

Input:	$\mathbf{s}'(t)$ -- a vector holding all the normalized sensory input arrays $\mathbf{S}'_{1,t} \dots \mathbf{S}'_{\Theta_{\sigma},t}$
Output:	$lnm_t = [\mathbf{LNM}_{(1, \gamma, t)}, \mathbf{LNM}_{(2, \gamma, t)}, \mathbf{LNM}_{(3, \gamma, t)}, \dots, \mathbf{LNM}_{(\Theta_{\sigma}, \gamma, t)}]$ The local navigation maps \mathbf{LNM} of each sensory system which best match the corresponding sensory input array $\mathbf{S}'_{\sigma,t}$, and then are updated to \mathbf{LNM}' with the actual sensory information conveyed by $\mathbf{S}'_{\sigma,t}$ (for simulation compatibility issues renaming $\mathbf{LNM}'_{(\sigma, \gamma, t)}$ to $\mathbf{LNM}_{(\sigma, \gamma, t)}$ again)
Description:	-The normalized sensory input arrays are best matched with stored navigation maps in each sensory system (called a local navigation map \mathbf{LNM}). -The best matched \mathbf{LNM} s in each sensory system are then updated with the actual sensory information from the sensory input for that sensory system. -The updated best matched \mathbf{LNM} is stored locally in that Input Sensory Vectors Shaping Module for matching against future sensory inputs. -The updated best matched \mathbf{LNM} is output from this module; the best matched \mathbf{LNM} s from each Input Sensory Vectors Shaping Module together make up vector lnm_t --the effective output.

	-Applies to CCA5, CCA6 and CCA7 versions of the architecture
--	--

Table A5. Summary of the Operation of the Input Sensory Vectors Associations Modules per Equations (12) – (23)

A.3 Data Structures in the CCA6

This section of Appendix A applies to the CCA5 (shown in Figure 1), the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

This section reviews a number of data structures used in the CCA6 architecture, most of them being compatible with the navigation map data structure. Equation (14) defines the local navigation map **LNM** as an array of four (or more) dimensions. There are three spatial dimensions (m,n,o in the definition representing x,y,z) as well as an extra dimension p for non-spatial information such storing which features belong to which objects, for storing meta-data, and so on.

Local navigation maps (i.e., stored locally in each sensory Input Sensory Vectors Association Module) were defined above in (14). Multisensory navigation maps **NM**, instinctive primitive navigation maps **IPM**, and learned primitive navigation maps **LPM** are similarly defined in (24). The multisensory navigation maps **NM** are stored in the Causal Memory Module (Figure 1) and contain visual, auditory, olfactory, etc. sensory features unlike the local navigation maps which have features from only one sensory system. The instinctive primitive navigation maps **IPM** and the learned primitive navigation maps **LPM** have the same dimensional structure as other navigation maps, but they contain largely only procedures to perform on other navigation maps.

Equation (26) defines *all_LNMs_t* as holding all the local navigation maps **LNM** in all the different Input Sensory Vectors Association Modules. Recall from above that *all_maps_{σ,t}* holds all the local navigation maps **LNM** within a *given* Input Sensory Vectors Association Module. In (27) *all_NMs_t* are defined as holding all the multisensory navigation maps **NM** in the Causal Memory Module (Figure 1). In (28) *all_IPMs_t* are defined as holding all the instinctive primitive navigation maps **IPM** (or “instinctive primitives”—the procedures that are included with the architecture) in the Instinctive Primitives Module (Figure 1). In (29) *all_LPMs_t* are defined as holding all the learned primitive navigation maps **LPM** (or “learned primitives”—the procedures that are learned by the architecture) in the Learned Primitives Module (Figure 1). And in (30) *all_navmaps_t* are defined as holding all of these preceding navigation maps, i.e., [*all_LNMs_t*, *all_NMs_t*, *all_IPMs_t*, *all_LPMs_t*]. (The vector *all_navmaps_t* does not hold the entirety of navigation maps in the CCA6 architecture as there are a number of other specialized navigation maps, particularly in the Sequential/Error Correcting Module.)

In (31–33) an addressing protocol is defined to address any particular cell within any particular navigation map within *all_navmaps_t*. For example, $\chi_{\text{modcode}=\text{*Causal_Memory_Mod*}, \text{mapno}=3456, \text{x}=2, \text{y}=3, \text{z}=4}$ is cell x=2,y=3,z=4 in map number 3456 in the Causal Memory Module.

In (34) a “*feature*” is defined as some arbitrary real number representing a feature modality and value. For example, *feature*_{3, ($\chi_{\text{modcode}=\text{*Causal_Memory_Mod*}, \text{mapno}=3456, \text{x}=2, \text{y}=3, \text{z}=4}$)} would be *feature* number 3 in the cell x=2,y=3,z=4 in map number 3456 in the Causal Memory Module. Its value, for example, could represent a visual line. In (35) similarly a “*procedure*” is defined as some arbitrary real number

representing a procedure modality and value. For example, $procedure_{3,(\chi_{modcode}=\text{Causal_Memory_Mod*}, mapno=3456, x=2, y=3, z=4)}$ would be *procedure* number 3 in the cell $x=2, y=3, z=4$ in map number 3456 in the Causal Memory Module. Its value, for example, could represent a *procedure* (and required sub-procedures too; the value can have an unlimited number of digits) to move forward. In (36) a *linkaddress* is defined as χ' , i.e., pointing to χ' which is some cell location in some navigation map in the architecture. The linkaddress provides a link between one cell in one map to another cell, possibly in the same map but often in a different navigation map. (Note: For stylistic reasons, in some places in the text “*features*” may be written which should be taken as the same variable as “*feature*”.)

In (38) the equation defines $cellfeatures_{\chi, t}$ as all the features within a given cell χ . In (39) the equation defines $cellprocedures_{\chi, t}$ as all the procedures within a given cell χ . In (40) the equation defines $linkaddresses_{\chi, t}$ as all the linkaddresses within a given cell χ . In (41) the equation defines $cellvalues_{\chi, t}$ as all the values—the features, the procedures, the link addresses—held by a cell of a navigation map at location χ . The numbers in each cell can represent any collection of low-level sensory features, higher-level sensory features, procedures, and links.

As shown above, *all_navmaps* represents all the navigation maps in the architecture. In (41) it is seen that the value of a cell in one of the navigation maps in *all_navmaps* are its $cellvalues_{\chi, t}$, i.e., the features, the procedures and the linkaddresses that the cell contains. Equation (42) shows that the pseudocode $link(\chi, t)$ will return all the links that a cell at address χ contains.

These equations remain largely unchanged in the CCA6 and CCA7 architectures from the prior CCA5 architecture.

$$NM_{mapno} \in R^{mxnxoxp}, IPM_{mapno} \in R^{mxnxoxp}, LPM_{mapno} \in R^{mxnxoxp} \quad (24)$$

$$\Theta_NM = \text{total used NM's} \in N, \Theta_IPM = \text{total used IPM's} \in N, \Theta_LPM = \text{total used LPM's} \in N \quad (25)$$

$$all_LNMs_t = [all_maps_{1,t}, all_maps_{2,t}, all_maps_{3,t}, \dots, all_maps_{\Theta_NM,t}] \quad (26)$$

$$all_NMs_t = [NM_{1,t}, NM_{2,t}, NM_{3,t}, \dots, NM_{\Theta_NM,t}] \quad (27)$$

$$all_IPMs_t = [IPM_{1,t}, IPM_{2,t}, IPM_{3,t}, \dots, IPM_{\Theta_IPM,t}] \quad (28)$$

$$all_LPMs_t = [LPM_{1,t}, LPM_{2,t}, LPM_{3,t}, \dots, LPM_{\Theta_LPM,t}] \quad (29)$$

$$all_navmaps_t = [all_LNMs_t, all_NMs_t, all_IPMs_t, all_LPMs_t] \quad (30)$$

$$modcode = \text{module identification code} \in N \quad (31)$$

$$mapcode = [modcode, mapno] \quad (32)$$

$$\chi = [mapcode, x, y, z] \quad (33)$$

$$feature \in R \quad (34)$$

$$procedure \in R \quad (35)$$

$$linkaddress \chi' \in \chi \quad (36)$$

$\Phi_feature$ = last *feature* contained by a cell, $\Phi_procedure$ = last *procedure* contained by a cell,
 Φ_x = last x (i.e., address) contained by a cell (37)

$$cellfeatures_{x,t} = [feature_{1,t}, feature_{2,t}, feature_{3,t}, \dots, feature_{\Phi_feature,t}] \quad (38)$$

$$cellprocedures_{x,t} = [procedure_{1,t}, procedure_{2,t}, procedure_{3,t}, \dots, procedure_{\Phi_procedure,t}] \quad (39)$$

$$linkaddresses_{x,t} = [x_{1,t}, x_{2,t}, x_{3,t}, \dots, x_{\Phi_x,t}] \quad (40)$$

$$cellvalues_{x,t} = [cellfeatures_{x,t}, cellprocedures_{x,t}, linkaddresses_{x,t}] \quad (41)$$

$$cellvalues_{x,t} = all_navmaps_{x,t} \quad (42)$$

$$linkaddresses_{x,t} = link(x,t) \quad (43)$$

LNM	<p>local navigation map – one sensory system maps to any local navigation map</p> <p>different set of LNMs for each sensory system</p> <p>each set of LNMs stored in that particular sensory Input Sensory Vectors Association Module (Figure 1)</p> <p>array structure of dimensions m,n,o,p (x,y,z and non-spatial p dimension)</p>
NM	<p>multisensory navigation map – different sensory system features can be written to this map</p> <p>stored in the Causal Memory Module (Figure 1)</p> <p>array structure of dimensions m,n,o,p (x,y,z and non-spatial p dimension)</p>
IPM	<p>-instinctive primitive navigation map – “instinctive primitive”</p> <p>-primitives are procedures to perform on cells of other navigation maps</p> <p>-primitives are navigation maps mainly filled with procedures in their cells, but they can store features and linkaddresses in their cells as well</p> <p>-these primitives come with the architecture</p> <p>-stored in the Instinctive Primitives Module (Figure 1)</p> <p>-array structure of dimensions m,n,o,p (x,y,z and non-spatial p dimension)</p>
LPM	<p>-learned primitive navigation map – “learned primitive”</p> <p>-primitives are procedures to perform on cells of other navigation maps</p> <p>-primitives are navigation maps mainly filled with procedures in their cells, but they can store features and linkaddresses in their cells as well</p> <p>-these primitives are learned by the architecture</p> <p>-stored in the Learned Primitives Module (Figure 1)</p>

	-array structure of dimensions m,n,o,p (x,y,z and non-spatial p dimension)
<i>all_maps₁</i>	vector of all of the LNMs in $\sigma=1$, i.e., visual Input Sensory Vectors Association Module (note: 't' which indicates changes with time has been removed here and following entries for simplification purposes) (<i>all_maps₂</i> i.e., $\sigma=2$, is auditory module, and so on)
<i>all_LNMs</i>	vector of all of the LNMs in all of the Input Sensory Vectors Association Modules
<i>all_NMs</i>	vector of all of the NM's in the Causal Memory Module
<i>all_IPMs</i>	vector of all of the IPM's in the Instinctive Primitives Module
<i>all_LPMs</i>	vector of all of the LPM's in the Learned Primitives Module
<i>all_navmaps</i>	vector of all of the LNMs, NM's, IPM's, LPM's in the architecture
<i>mapcode</i>	[module identification code, map number] points to a particular navigation map among all the LNMs, NM's, IPM's, and LPM's in the architecture
χ	[mapcode, x, y, z] points to particular cell (x,y,z) in a particular navigation map ([module identification code, map number])
<i>feature</i> <i>features</i>	arbitrary real number representing a feature modality and value within a cell χ (Note: For stylistic reasons, in some places the text may write " <i>features</i> " which should be taken as the same variable as " <i>feature</i> ".)
<i>procedure</i>	arbitrary real number representing a procedure modality and value within a cell χ
linkaddress	address within a cell χ pointing to another cell (possibly in another navigation map) χ'
<i>cellfeatures_{χ}</i>	vector of all the features within a cell χ
<i>cellprocedures_{χ}</i>	vector of all the procedures within a cell χ
<i>linkaddresses_{χ}</i>	vector of all the linkaddress within a cell χ
<i>cellvalues_{χ}</i>	vector of all of the features, procedures and linkaddresses within a cell χ
<i>all_navmaps_{χ}</i>	vector of all of the features, procedures and linkaddresses within a cell χ note that the value of some cell χ in some cell in some navigation map (i.e., within the collection of <i>all_navmaps</i>) = <i>cellvalues_{χ}</i>

$\text{link}(\chi, t)$	pseudocode that returns all the linkaddresses within a cell χ , i.e., same value as <i>linkaddresses_{χ}</i>
------------------------	---

Table A6. Explanation of Symbols and Pseudocode in Equations (24) – (43)

Input:	<i>not applicable – definitions of data structures in this section</i>
Output:	<i>not applicable – definitions of data structures in this section</i>
Description:	<p>-In this section some of the key data structures utilized by the architecture are discussed.</p> <p>-These data structures are largely based on the “navigation map” which maps spatial features (e.g., are there pixels representing ground at this x,y,z coordinate?), potential procedures (e.g., do something with the data in this or other cells of the navigation map), and link addresses (e.g., possibly go to the cell in possibly another navigation map specified by the link address) at a particular x,y,z coordinate (“cell” or “cube”).</p> <p>-The architecture in conjunction with the navigation map data structure allows a solution to the classical binding problem.</p> <p>-Applies to CCA5, CCA6 and CCA7 versions of the architecture</p>

Table A7. Summary of the Operation of Equations (24) – (43)

A.4 The Sequential/Error Correcting Module

This section of Appendix A applies to the CCA5 (shown in Figure 1), the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

Consider the sensory inputs through the CCA6 architecture as they become temporally bound. The Sequential/Error Correcting Module plays a key role in temporally binding the sensory inputs.

Normalized sensory input arrays $\mathbf{S}'_{\sigma, t}$ represented by $s'(t)$ (from the output of the Input Sensory Vectors Shaping Modules (10, 11)) feed into the Sequential/Error Correcting Module. After processing these signals the Sequential/Error Correcting Module sends a “motion prediction vector” to the Navigation Module. The motion prediction vector allows changes in an object(s) in a navigation map to be represented much as other spatial features on a navigation map.

In (44) s'_{series_t} is a time series of the input sensory vector $s'(t)$. In (45) and (46) the visual and auditory sensory times series are extracted from s'_{series_t} and stored respectively as *visual_series_t* and *auditory_series_t*. Although it is possible to create motion prediction vectors for all the senses, in the current simulation this is only done so for the visual and auditory sensory systems.

In (47) the pseudocode `Sequential_Mod.visual_match()` matches *visual_series_t* with visual time series stored in the Sequential/Error Correcting Module. If there is a reasonable match with few differences, then the matched time series will be updated with the new information. If there is no close

enough match, then a new times series will be stored in the Sequential/Error Correcting Module. A motion prediction vector $\mathbf{visual_motion}_t$ is then computed from visual time series data (47). A similar process occurs in computing the motion prediction vector $\mathbf{auditory_motion}_t$ (48). These motion prediction vectors are then stored much like any spatial feature in a navigation map called the Vector Navigation Map \mathbf{VNM}''_t (50a, 50b). \mathbf{VNM}''_t is then propagated to the Navigation Module complex (Figure 1).

In (51) the pseudocode `Sequential_Mod.auditory_match_process()` extracts sound patterns from $\mathbf{auditory_series}_t$, and stores these patterns spatially in a navigation map \mathbf{AVNM}_t . Then \mathbf{AVNM}_t is propagated to the Navigation Module (Figure 1) or Navigation Module A (Figure 5).

These equations remain largely unchanged in the CCA6 and CCA7 architectures from the prior CCA5 architecture, other than distinguishing the two Navigation Modules as mentioned in the text.

$$\mathbf{s}'_series_t = [s'(t-3), s'(t-2), s'(t-1), s'(t)] \quad (44)$$

$$\mathbf{visual_series}_t = \text{Sequential_Mod.visual_inputs}(\mathbf{s}'_series_t) \quad (45)$$

$$\mathbf{auditory_series}_t = \text{Sequential_Mod.auditory_inputs}(\mathbf{s}'_series_t) \quad (46)$$

$$\mathbf{visual_motion}_t = \text{Sequential_Mod.visual_match}(\mathbf{visual_series}_t) \quad (47)$$

$$\mathbf{auditory_motion}_t = \text{Sequential_Mod.auditory_match}(\mathbf{auditory_series}_t) \quad (48)$$

$$\mathbf{VNM} \in \mathbb{R}^{m \times n \times o \times p}, \mathbf{AVNM} \in \mathbb{R}^{m \times n \times o \times p} \quad (49)$$

$$\mathbf{VNM}'_t = \mathbf{VNM}_t \cup \mathbf{visual_motion}_t \quad (50a)$$

$$\mathbf{VNM}''_t = \mathbf{VNM}'_t \cup \mathbf{auditory_motion}_t \quad (50b)$$

$$\mathbf{AVNM}_t = \text{Sequential_Mod.auditory_match_process}(\mathbf{auditory_series}_t) \quad ((51))$$

In the next section it will be seen that the Object Segmentation Gateway Module (Figure 1) will segment a sensory scene into objects of interest. The individual objects segmented in the sensory scene, as well as the entire scene itself treated as one composite object, will then trigger similar navigation maps in the Causal Memory Module to be retrieved and moved to the Navigation Module A. For example, a visual scene of a river, a rock and a leaf floating in the river, might be segmented in the river object, the rock object and the leaf object. In order to obtain the motion information about a segmented object, each segmented object navigation map must be sent to the Sequential/Error Correcting Module.

In (52) a navigation map called the Visual Segmented Navigation Map \mathbf{VSNM} is defined. $\mathbf{VSNM}_{i,t-3}$, $\mathbf{VSNM}_{i,t-2}$, $\mathbf{VSNM}_{i,t-1}$, and $\mathbf{VSNM}_{i,t}$ containing a visual segmented object on a navigation map at different time intervals, are sent from the Object Segmentation Gateway Module to the Sequential/Error Correcting Module where they are stored in vector $\mathbf{visual_segmented_series}_{i,t}$ (53). There may be several \mathbf{VSNM} 's produced for one sensory scene in the Object Segmentation Gateway Module, e.g., a sensory scene of a river with a rock and leaf floating in, it will produce a \mathbf{VSNM} for

the river, for the rock and for the leaf. Hence, the use of subscript i to refer to a particular \mathbf{VSNM}_i for the sensory scene in a cognitive cycle.

The same pseudocode used in (47) is used again in (54) but this time on *visual_segmented_series* _{i,t} and produces *visseg_motion* _{i,t} which is a motion prediction vector for the motion information, if any, in *visual_segmented_series* _{i,t} . This motion prediction vector is copied, like any spatial feature, onto the original navigation map $\mathbf{VSNM}_{i,t}$ and the updated Visual Segmented Navigation Map $\mathbf{VSNM}'_{i,t}$ is then sent back to the Object Segmentation Gateway Module/Navigation Module (55).

These equations remain largely unchanged in the CCA6 architecture from the prior CCA5 architecture, other than distinguishing the two Navigation Modules as mentioned in the text. This largely continues to apply to the CCA7 architecture.

$$\mathbf{VSNM} \in \mathbb{R}^{m \times n \times o \times p} \quad (52)$$

$$\text{visual_segmented_series}_{i,t} = [\mathbf{VSNM}_{i,t-3}, \mathbf{VSNM}_{i,t-2}, \mathbf{VSNM}_{i,t-1}, \text{ and } \mathbf{VSNM}_{i,t}] \quad (53)$$

$$\text{visseg_motion}_{i,t} = \text{Sequential_Mod.visual_match}(\text{visual_segmented_series}_{i,t}) \quad (54)$$

$$\mathbf{VSNM}'_{i,t} = \mathbf{VSNM}_{i,t} \cup \text{visseg_motion}_{i,t} \quad (55)$$

$s'(t)$	<p>-vector representing the normalized sensory input sensory arrays of the different sensory systems $\mathbf{S}'_{\sigma,t}$ produced by the Input Sensory Vectors Shaping Module (Figure 5)</p> <p>-sent to both Input Sensory Vectors Association Modules and to the Sequential/Error Correcting Module (Figure 1 or Figure 5 5)</p>
$s'(t-1)$	$s'(t)$ value in the previous cognitive cycle, i.e., $t-1$
s'_{series_t}	<p>a time series of s' at time t, time $t-1$, time $t-2$ (two cognitive cycles ago) and time $t-3$</p> <p>$= [s'(t-3), s'(t-2), s'(t-1), s'(t)]$</p>
<p>Sequential_Mod.visual_inputs(s'_{series_t})</p> <p>→ <i>visual_series</i>_{t}</p>	<p>-pseudocode for an algorithm that extracts the visual sensory normalized inputs, i.e., $\mathbf{S}'_{1,t}$ ($\sigma=1$ for visual system) from $s'(t)$</p> <p>-produces <i>visual_series</i>_{t} as its output, which essentially is $[\mathbf{S}'_{1,t}, \mathbf{S}'_{1,t-1}, \mathbf{S}'_{1,t-2}, \mathbf{S}'_{1,t-3}]$</p>
<p>Sequential_Mod.auditory_inputs(s'_{series_t})</p> <p>→ <i>auditory_series</i>_{t}</p>	<p>-pseudocode for an algorithm that extracts the auditory sensory normalized inputs, i.e., $\mathbf{S}'_{2,t}$ ($\sigma=2$ for auditory system) from $s'(t)$</p> <p>-produces <i>auditory_series</i>_{t} as its output, which is $[\mathbf{S}'_{2,t}, \mathbf{S}'_{2,t-1}, \mathbf{S}'_{2,t-2}, \mathbf{S}'_{2,t-3}]$</p>
Sequential_Mod.visual_match(<i>visual_series</i> _{t})	-pseudocode for an algorithm that matches <i>visual_series</i> _{t} with visual time series stored in the Sequential/Error Correcting Module (Figure 1 or Figure 5 5)

<p>→ <i>visual_motion_t</i></p>	<p>-a best match is chosen (or if no matches close enough then <i>visual_series_t</i> used itself as the best match</p> <p>-the best match is then updated from <i>visual_series_t</i></p> <p>-the updated best match is stored in the Sequential/Error Correcting Module for future matching</p> <p>-the updated best match is then transformed into a motion prediction vector (i.e., showing and predicting the motion of the object) is output as <i>visual_motion_t</i></p>
<p>Sequential_Mod.auditory_match(<i>auditory_series_t</i>)</p> <p>→ <i>auditory_motion_t</i></p>	<p>-pseudocode for an algorithm that matches <i>auditory_series_t</i> with auditory time series stored in the Sequential/Error Correcting Module (Figure 1 or Figure 5 5)</p> <p>-a best match is chosen (or if no matches close enough then <i>auditory_series_t</i> used itself as the best match</p> <p>-the best match is then updated from <i>auditory_series_t</i></p> <p>-the updated best match is stored in the Sequential/Error Correcting Module for future matching</p> <p>-the updated best match is then transformed into a motion prediction vector (i.e., showing and predicting the motion of the object) and is output as <i>auditory_motion_t</i></p>
VNM	a “Vector Navigation Map” is just another ordinary navigation map used to store the motion prediction vectors
AVNM	an “Audio Vector Navigation Map” is just another ordinary navigation map used to store more detailed motion prediction vectors about the sound patterns, useful for advanced analysis of sound patterns in perceiving the environment and for language
<p>$VNM_t \cup visual_motion_t$</p> <p>→ VNM'_t</p>	<p>the <i>visual_motion_t</i> motion prediction vector is copied to, i.e., stored, in the Vector Navigation Map VNM just like any other spatial feature – binding of temporal features as a spatial feature occurs here</p> <p>VNM'_t is the updated VNM (essentially a new navigation map has <i>visual_motion_t</i> copied to it)</p>
<p>$VNM'_t \cup auditory_motion_t$</p> <p>→ VNM''_t</p>	<p>-the <i>auditory_motion_t</i> motion prediction vector is copied to, i.e., stored, in the Vector Navigation Map VNM' just like any other spatial feature – binding of temporal features as a spatial feature occurs here again</p> <p>-VNM''_t is the updated VNM' (essentially VNM''_t is a navigation map which has <i>visual_motion_t</i> and <i>auditory_motion_t</i> copied to it – vectors respectively showing the previous and predicted motion of an object based on visual sensory inputs and based on auditory sensory inputs)</p> <p>-(note: in the simulation at present if there are different <i>visual_motion_t</i> and <i>auditory_motion_t</i> vectors, i.e., pointing in different directions, then the better quality result is used as a sole result in upstream algorithms in the Navigation Module; in the future more sophisticated algorithms can be used to take advantage of these motion prediction vectors)</p>
<p>Sequential_Mod.auditory_match_process(<i>auditory_series_t</i>)</p>	<p>-pseudocode for an algorithm that matches <i>auditory_series_t</i> with auditory time series stored in the Sequential/Error Correcting Module (Figure 1 or Figure 5 5)</p>

<p>→ → $AVNM_t$</p>	<p>-a best match is chosen (or if no matches close enough then <i>auditory_series_t</i> used itself as the best match</p> <p>-the best match is then updated from <i>auditory_series_t</i></p> <p>-the updated best match is stored in the Sequential/Error Correcting Module for future matching</p> <p>-the updated best match is then transformed into a complex advanced motion prediction vector (i.e., showing in more detail the pattern of the auditory sensations) and is output as $AVNM_t$ (which as described above is an “Audio Vector Navigation Map”)</p>
<p>$VSNM_{i,t}$</p>	<p>-a “Visual Segmentation Navigation Map” $VSNM$ which is an ordinary navigation map with visually segmented information stored on it</p> <p>-the Object Segmentation Gateway Module (Figure 1 or Figure 5 5) will “segment” a sensory scene into objects of interest (described in more detail in the following section)</p> <p>-each segmented object is treated as a separate navigation map (there is also a navigation map of all the objects together on the navigation map)</p> <p>-the subscript i refers to multiple $VSNM$’s that can be created for a given sensory scene</p> <p>-the subscript t refers to time since the values change each cognitive cycle</p> <p>-it is useful to calculate motion prediction vectors, if they exist (often they will not) for each segmented object (e.g., if there is a river object, a rock object and leaf floating in the river object, then it is very useful to segment the sensory scene into these objects (i.e., river, rock and leaf) and a motion prediction vector to show the motion of the leaf is very useful – thus there will be $VSNM_{1,t}$ for river, $VSNM_{2,t}$ for rock and $VSNM_{3,t}$ for leaf, in this example)</p> <p>-only visual motion is calculated for segmented objects (auditory motion and motion of other senses is not calculated in the current simulation, but could be in the future, for example, the motion of a radar signal in a future embodiment)</p>
<p><i>visual_segmented_series_{i,t}</i></p>	<p>-a time series of the current $VSNM_{i,t}$ and $VSNM_i$ from one cognitive cycle ago (i.e., t-1), two cycles ago (i.e., t-2) and three cycles ago (i.e., t-3)</p> <p>= [$VSNM_{i,t-3}$, $VSNM_{i,t-2}$, $VSNM_{i,t-1}$, and $VSNM_{i,t}$]</p> <p>-if there were three $VSNM$’s produced from scene (such as the example of the river with the rock and the leaf floating in it) then there would be three different <i>visual_segmented_series_{i,t}</i> produced, with $i=1$, $i=2$, and $i=3$</p>
<p>Sequential_Mod.visual_match(<i>visual_segmented_series_{i,t}</i>)</p> <p>→ <i>visseg_motion_{i,t}</i></p>	<p>-pseudocode for an algorithm that matches <i>visual_segmented_series_{i,t}</i> with visual time series stored in the Sequential/Error Correcting Module (Figure 1 or Figure 5 5)</p> <p>-a best match is chosen (or if no matches close enough then <i>visual_segmented_series_{i,t}</i> is used itself as the best match</p> <p>-the best match is then updated from <i>visual_segmented_series_{i,t}</i></p> <p>-the updated best match is stored in the Sequential/Error Correcting Module for future matching</p>

	<p>-the updated best match is then transformed into a motion prediction vector (i.e., showing and predicting the motion of the object) and is output as <i>visseg_motion_{i,t}</i></p>
<p>$\text{VSNM}_{i,t} \cup \text{visseg_motion}_{i,t}$</p> <p>$\rightarrow \text{VSNM}'_{i,t}$</p>	<p>-$\text{VSNM}_{i,t}$ was previously received from the Object Segmentation Gateway Module (Figure 1 or Figure 5 5) containing the spatial information about an object (also received and stored were [$\text{VSNM}_{i,t-3}$, $\text{VSNM}_{i,t-2}$, $\text{VSNM}_{i,t-1}$, and $\text{VSNM}_{i,t}$] so that a motion prediction vector could be calculated)</p> <p>-in this step the motion prediction vector <i>visseg_motion_{i,t}</i> about the object (e.g., perhaps the leaf in the example above) in this VSNM_t navigation map is copied to the $\text{VSNM}_{i,t}$ navigation map—binding of temporal features as a spatial feature occurs here for this segmented object</p> <p>-$\text{VSNM}'_{i,t}$ is produced in this operation (i.e., the original $\text{VSNM}_{i,t}$ plus the motion prediction vector (<i>visseg_motion_{i,t}</i>) and is sent back to Object Segmentation Gateway Module/Navigation Module (Figure 1 or Figure 5 5)</p> <p>-if there were three VSNM's produced from scene (such as the example of the river with the rock and the leaf floating in it) then there would be three different $\text{VSNM}'_{i,t}$ produced and returned to the Object Segmentation Gateway Module/Navigation Module (Figure 1 or Figure 5 5), with $i=1$, $i=2$, and $i=3$</p>

Table A8. Explanation of Symbols and Pseudocode in Equations (44) – (55)

Input:	<p>- $\mathbf{s}^*(t)$ is a vector representing all the normalized sensory input arrays $\mathbf{S}^*_{1,t} \dots \mathbf{S}^*_{\Theta_{\sigma,t}}$</p> <p>($\mathbf{s}^*(t-1)$, $\mathbf{s}^*(t-2)$, $\mathbf{s}^*(t-3)$ also from $t-1$, $t-2$ and $t-3$ previous cognitive cycles are stored so that a time series can be processed)</p> <p>-$\text{VSNM}_{i,t}$ is one of several VSNM navigation maps propagated from the Object Segmentation Gateway Module containing visual segments (hence the name VSNM), i.e., segments of the sensory scene recognized as distinct objects</p> <p>-e.g., if the sensory scene was a river with a rock and a leaf floating down a river, a $\text{VSNM}_{1,t}$ could contain the river, a $\text{VSNM}_{2,t}$ could contain the rock, and a $\text{VSNM}_{3,t}$ could contain the leaf – they are being sent to the Sequential/Error Correcting Module for temporal binding of motion, i.e., for insertion of a motion prediction vector if the object is moving</p>
Output:	<p>-VNM'_t is a navigation map binding visual motion and auditory motion of the sensory scene) propagated to the Navigation Module A complex (Figure 1 or Figure 5 5)</p> <p>-AVNM_t is a navigation map binding advanced auditory patterns propagated to the Navigation Module A complex (Figure 1 or Figure 5 5) (useful for auditory analysis as in better recognition of the environment and language)</p> <p>-$\text{VSNM}'_{i,t}$ is the original $\text{VSNM}_{i,t}$ from the Object Segmentation Gateway Module with a motion prediction vector added if the object was moving (or else unchanged if there was no movement), and it is then returned to the Object Segmentation Gateway Module/Navigation Module A (Figure 1 or Figure 5 5)</p>
Description:	<p>-If there is movement in a sensory scene rather than requiring thirty versions of the sensory scene per second, only a single version is required but a motion prediction vector is added to show motion that has occurred and is still predicted to occur.</p>

	<p>-\mathbf{VNM}'_t is a navigation map produced with motion prediction vectors showing any motion with regard to overall visual and sound features in the sensory scene.</p> <p>-\mathbf{AVNM}_t is a navigation map produced with multiple motion prediction vectors showing more detailed and advanced sound patterns useful for environment recognition and spoken language.</p> <p>-$\mathbf{VSNM}'_{i,t}$ is a navigation map for a given object (or “segment”) of the sensory scene with a motion prediction vector added if there is motion of the object. There can be several (i.e., $i=1, i=2$, and so on) $\mathbf{VSNM}'_{i,t}$ navigation maps for any given scene.</p> <p>-\mathbf{VNM}'_t, \mathbf{AVNM}_t, and $\mathbf{VSNM}'_{i,t}$ are sent to the Object Segmentation Gateway Module/Navigation Module (Figure 1 or Figure 5 5).</p> <p>-This module allows a solution to the temporal binding.</p> <p>-Although the examples such as river, rock and leaf floating in it, are very concrete ones, there can be binding of motion of more abstract concepts by the same mechanisms described above.</p> <p>-Applies to CCA5, CCA6 and CCA7 versions of the architecture</p>
--	--

Table A9. Summary of the Operation of the Sequential/Error Correcting Module per Equations (44) – (55)

A.5 The Object Segmentation Gateway Module

This section of Appendix A applies to the CCA5 (shown in Figure 1), the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

At this point the sensory inputs have been transformed into best matching and updated visual, auditory, and olfactory local navigation maps (represented by the vector \mathbf{lnm}_t). The local navigation maps \mathbf{LNMs} are propagated to the Object Segmentation Gateway Module. The complex of three tightly connected modules—the Object Segmentation Gateway Module, the Navigation Module and the Causal Memory Module will transform the input sensory data into a Working Navigation Map \mathbf{WNM} upon which instinctive and learned primitives (i.e., essentially small algorithms) can act and possibly produce an action output.

The Object Segmentation Gateway Module will attempt to segment each sensory scene into different objects. In the current version of the CCA6 it is only performed visually—the module attempts to recognize coherent visual shapes in the sensory scene. (In theory, this could be done with multiple senses, and may be done in future versions of the architecture.)

Continuing the example given above of a sensory scene with only visual stimuli— a river with a rock and leaf floating in the river, the leaf, the rocks and the river are recognized by the Object Segmentation Gateway Module and segmented as separate objects each on their own \mathbf{VSNM} navigation map (while at the same time keeping the entire scene and all objects on another navigation map). There will be an attempt to calculate motion prediction vectors for each different object’s navigation map, but only the leaf will have a motion prediction vector since its position is changing in this example. Note that the use of labels such as ‘river’, ‘rocks’, ‘leaf’ is for the benefit of the reader. The CCA6 does not have a full English language implemented at this time, and uses its own internal labels. As well, current recognition is from a catalog of objects and simulated—the architecture has little deep understanding of the objects.

The “Visual Segmentation Navigation Map” $\mathbf{VSNM}_{i,t}$ (52) is an ordinary navigation map with visually segmented information stored on it, i.e., each object such as the leaf floating in the river, the

rocks, and the river of the example get put onto a separate \mathbf{VSNM}_i navigation map. For example, $\mathbf{VSNM}_{1,t}$ for river, $\mathbf{VSNM}_{2,t}$ for one of the group of rocks, and perhaps $\mathbf{VSNM}_{3,t}$ for the leaf. (There is a subscript t since the values of these navigation maps change each cognitive cycle.)

The pseudocode/algorithm `Object_Seg_Mod.visualsegment` in equation (61) takes the best-matching visual input sensory local navigation map $\mathbf{LNM}_{(1, \gamma, t)}$ and segments it into whatever objects it can find in its local memory stores or that meet certain algorithmic criteria (e.g., pixels separate from other pixels, and so on). The navigation maps $\mathbf{VSNM}_{t,i=1..\Theta_i}$ (e.g., in the example above of the river, group or rocks and leaf producing $\mathbf{VSNM}_{1,t}$ for river, $\mathbf{VSNM}_{2,t}$ for one of the group of rocks, and perhaps $\mathbf{VSNM}_{3,t}$ for the leaf) are produced as a result. These navigation maps $\mathbf{VSNM}_{t,i=1..\Theta_i}$ are sent to Sequential/Error Correcting Module where according to equations (52–55) attempts are made to see if there is motion occurring (via time series of \mathbf{VSNM} 's from different cognitive cycles) and if so a motion prediction vector is bound onto the particular \mathbf{VSNM} navigation map, which is returned back to the Object Segmentation Gateway.

One of the arguments to pseudocode/algorithm `Object_Seg_Mod.visualsegment` (61) is as noted above the best-matching visual input sensory local navigation map $\mathbf{LNM}_{(1, \gamma, t)}$. This \mathbf{LNM} is actually transmitted in parallel to the Object Segmentation Gateway Module along with the other \mathbf{LNM} s from other sensory system. Having the different sensory systems' best-matching \mathbf{LNM} s via vector \mathbf{lnm} , and in (56) it is trivial to extract $\mathbf{LNM}_{(1, \gamma, t)}$. Another argument is the contextual value **CONTEXT**, which actually is a navigation map, and will help influence which objects are detected. In (57–59) it is seen to be set to the value of the previous cognitive cycle's Working Navigation Map \mathbf{WNM}' , which effectively is the previous cognitive cycle's action taken or intermediate results. The contextual value will help influence which objects are detected. The third argument to (61) is \mathbf{VNM}''_t . As shown above in the Sequential/Error Correcting Module, \mathbf{VNM}''_t shows the main visual motion and sound motion of the overall sensory scene. The vector navigation map \mathbf{VNM}''_t helps influence segmentation in (61) by different motions and sound production. In (62) $\mathbf{LNM}_{(1, \gamma, t)}$ is updated to $\mathbf{LNM}'_{(1, \gamma, t)}$ with the information (i.e., motion prediction vectors, which does include visual and sensory motion of the main object in the scene) from \mathbf{VNM}''_t .

These equations remain largely unchanged in the CCA6 architecture from the prior CCA5 architecture.

$$\mathbf{LNM}_{(1, \gamma, t)} = \mathbf{lnm}_t[0] \quad (56)$$

$$\mathbf{CONTEXT} = \in \mathbb{R}^{m \times n \times o \times p} \quad (57)$$

$$\mathbf{WNM}' = \in \mathbb{R}^{m \times n \times o \times p} \quad (58)$$

$$\mathbf{CONTEXT}_t = \mathbf{WNM}'_{t-1} \quad (59)$$

$$\Theta_i = \text{total objects segmented in this sensory scene} \in \mathbb{N} \quad (60)$$

$$\mathbf{VSNM}_{t,i=1..\Theta_i} = \text{Object_Seg_Mod.visualsegment}(\mathbf{LNM}_{(1, \gamma, t)}, \mathbf{CONTEXT}_t, \mathbf{VNM}''_t) \quad (61)$$

$$\mathbf{LNM}'_{(1, \gamma, t)} = \mathbf{LNM}_{(1, \gamma, t)} \cup \mathbf{VNM}''_t \quad (62)$$

$\mathbf{LNM}_{(\sigma, \gamma, t)}$ Equation (18) described in an earlier section	<p>the local navigation map LNM stored in the Input Sensory Vectors Association Module σ with map number γ which best matches in the incoming σ sensory inputs (e.g., if $\sigma = 1$ then it would be visual sensory inputs)</p> <p>e.g., if the visual sensory inputs best match the local navigation map #3456 in the visual module of the Input Sensory Vectors Association Modules, then at that moment $\mathbf{LNM}_{(1, \gamma, t)}$ would be navigation map #3456 in the visual module</p> <p>-Applies to CCA5, CCA6 and CCA7 versions of the architecture</p>
$\mathbf{LNM}_{(\sigma, \gamma, t)} \cup \mathbf{S}'_{\sigma, t}$ $\rightarrow \mathbf{LNM}$ Equation (21) described previously	<p>as shown previously:</p> <p>update $\mathbf{LNM}_{(\sigma, \gamma, t)}$ with any new information in input sensory signal $\mathbf{S}'_{\sigma, t}$ (i.e., copy $\mathbf{S}'_{\sigma, t}$ onto $\mathbf{LNM}_{(\sigma, \gamma, t)}$) thereby creating an updated $\mathbf{LNM}_{(\sigma, \gamma, t)}$</p>
$\mathbf{LNM}_{(\sigma, new_map, t)} \cup \mathbf{S}'_{\sigma, t}$ $\rightarrow \mathbf{LNM}$ Equation (22) described previously	<p>OR</p> <p>copy $\mathbf{S}'_{\sigma, t}$ onto an empty $\mathbf{LNM}_{(\sigma, new_map, t)}$ thereby creating an updated $\mathbf{LNM}_{(\sigma, \gamma, t)}$</p>
\mathbf{lnm}_t Equation (23) described previously	<p>i.e., a vector holding $[\mathbf{LNM}_{(1, \gamma, t)}, \mathbf{LNM}_{(2, \gamma, t)}, \mathbf{LNM}_{(3, \gamma, t)}, \dots, \mathbf{LNM}_{(O_\sigma, \gamma, t)}]$ -- the local navigation maps LNM of each sensory system which best match the corresponding sensory input array $\mathbf{S}'_{\sigma, t}$ and then are updated to LNM with the actual sensory information conveyed by $\mathbf{S}'_{\sigma, t}$</p>
$\mathbf{LNM}_{(1, \gamma, t)}$	<p>from the definition of \mathbf{lnm}_t this is just the first member of \mathbf{lnm}_t which is $\mathbf{lnm}_t[0]$ (indexing starts from 0 in this array) which is the LNM for the visual sensory inputs</p>
WNM'	<p>-Working Navigation Map</p> <p>-the current Working Navigation Map \mathbf{WNM}'_t is the navigation map which the Navigation Module A focuses its attention on, i.e., applies operations on to make a decision to take some sort or no action</p> <p>-as will be seen in the following sections, a navigation map can be assigned as being the \mathbf{WNM}'_t for that cognitive cycle, and then a different one in the next cognitive cycle, and so on, depending on the sensory information being processed and the results obtained</p>
CONTEXT	<p>-a normal navigation map (i.e., same dimensions as the other navigation maps in the architecture) used to hold contextual information which help influence which objects are detected</p> <p>-at present it is set to the value of the previous (i.e., in the previous cognitive cycle) Working Navigation Map \mathbf{WNM}'_{t-1}</p>
VNM Equation (49) described previously	<p>a “Vector Navigation Map” is just another ordinary navigation map used to store the motion prediction vectors</p>

<p>VNM''_t</p> <p>Equations (50a, 50b)</p> <p>described previously</p>	<p>as described previously in the Sequential/Error Correcting Module:</p> <ul style="list-style-type: none"> -essentially VNM''_t is a navigation map which has visual_motion_t and auditory_motion_t copied to it – vectors respectively showing the previous and predicted motion of an object based on visual sensory inputs and based on auditory sensory inputs -tends to refer to motion of the overall scene or major object rather than the segmented objects in the sensory scene
<p>VSNM_{i,t} → VSNM'_{i,t}</p> <p>(Note: VSNM_{i,t} is first created in the Object Segmentation Gateway Module, and then updated to VSNM'_{i,t} in the Sequential/Error Correcting Module)</p>	<ul style="list-style-type: none"> -a “Visual Segmentation Navigation Map” VSNM which is an ordinary navigation map with visually segmented information stored on it -the Object Segmentation Gateway Module will “segment” a sensory scene into objects of interest -each segmented object is treated as a separate navigation map (there is also a navigation map of all the objects together on the navigation map) -the subscript <i>i</i> refers to multiple VSNM’s that can be created for a given sensory scene -the subscript <i>t</i> refers to time since the values change each cognitive cycle -it is useful to calculate motion prediction vectors, if they exist (often they will not) for each segmented object (e.g., if there is a river object, a rock object and leaf floating in the river object, then it is very useful to segment the sensory scene into these objects (i.e., river, rock and leaf) and a motion prediction vector to show the motion of the leaf is very useful – thus there will be VSNM_{1,t} for river, VSNM_{2,t} for rock and VSNM_{3,t} for leaf, in this example) -continuing this example: VSNM_{1,t} for river, VSNM_{2,t} for rock and VSNM_{3,t} for leaf will be propagated to the Sequential/Error Correcting Module for computation of motion prediction vectors for each of these VSNM’s -continuing this example: a motion prediction vector is computed and stored on VSNM_{3,t} for leaf, but VSNM_{1,t} for river, VSNM_{2,t} for rock did not have enough motion for such vectors and are unchanged -continuing this example: VSNM'_{1,t} for river (unchanged), VSNM'_{2,t} for rock (unchanged) and VSNM'_{3,t} for leaf (motion prediction vector added) are then returned back to the Object Segmentation Gateway Module
<p>VSNM_{i=1..θ_i}</p>	<ul style="list-style-type: none"> -all the Visual Segmentation Navigation Maps VSNM’s created for a sensory scene -for example, in the example just given there would be VSNM_{1,t} for river, VSNM_{2,t} for rock and VSNM_{3,t} for leaf
<p>Object_Seg_Mod.visualsegment(LNM_(1, γ, n), CONTEXT, VNM''_t)</p>	<p>-pseudocode for an algorithm that takes the best-matching visual input sensory local navigation map LNM_(1, γ, n) and segments it into whatever objects it can find</p>

<p>→ $\mathbf{VSNM}_{t,i=1.. \Theta_i}$</p>	<p>in its local memory stores or that meet certain algorithmic criteria (e.g., pixels separate from other pixels, and so on)</p> <p>-currently only segments via visual features (although one of the arguments \mathbf{VNM}''_t can contain information about auditory motion)</p> <p>-arguments were described above:</p> <p>$\mathbf{LNM}_{(1, \gamma, t)}$: the best-matching visual input sensory local navigation map</p> <p>$\mathbf{CONTEXT}_t$: to hold contextual information which help influence which objects are detected</p> <p>\mathbf{VNM}''_t : auditory and visual motion of the overall scene or major object</p> <p>-produces $\mathbf{VSNM}_{t,i=1.. \Theta_i}$ which are all the Visual Segmentation Navigation Maps \mathbf{VSNM}'s created for a sensory scene</p>
<p>$\mathbf{LNM}'_{(1, \gamma, t)}$</p>	<p>$\mathbf{LNM}'_{(1, \gamma, t)}$ is the updated best-matching visual sensory Local Navigation Map</p> <p>$\mathbf{LNM}_{(1, \gamma, t)}$ further updated with any visual or auditory motion prediction vectors from \mathbf{VNM}''_t</p>

Table A10. Explanation of Symbols and Pseudocode in Equations (56) – (62)

<p>Initial Input:</p>	<p>-\mathbf{AVNM}_t : a navigation map binding advanced auditory patterns; used in the next section</p> <p>-$\mathbf{LNM}_{(1, \gamma, t)}$: the best-matching visual input sensory local navigation map; from the Input Sensory Vectors Association Module will be segmented for objects in the sensory scene</p> <p>-other sensory system \mathbf{LNM}s (represented by \mathbf{lnm}_t); used in the next section</p> <p>-\mathbf{WNM}'_t : Working Navigation Map from the Navigation Module; stored and used as \mathbf{WNM}'_{t-1} to produce a value for $\mathbf{CONTEXT}_t$ which holds contextual information which helps influence which objects are detected</p> <p>-\mathbf{VNM}''_t : auditory and visual motion of the overall scene or major object; from the Sequential/Error Correcting Module</p>
<p>Initial Output:</p>	<p>-produces $\mathbf{VSNM}_{t,i=1.. \Theta_i}$ which are all the Visual Segmentation Navigation Maps \mathbf{VSNM}'s created for a sensory scene; sent to the Sequential/Error Correcting Module</p>
<p>Final Input:</p>	<p>-$\mathbf{VSNM}'_{t,i=1.. \Theta_i}$: the Sequential/Error Correcting Module will add where indicated a motion prediction vector to the \mathbf{VSNM}'s it receives if there is motion of the object, and then send the $\mathbf{VSNM}'_{t,i=1.. \Theta_i}$ (i.e., all the \mathbf{VSNM}'s) back to the Object Segmentation Gateway Module</p> <p>-e.g., in the example above: : $\mathbf{VSNM}'_{1,t}$ for river (unchanged), $\mathbf{VSNM}'_{2,t}$ for rock (unchanged) and $\mathbf{VSNM}'_{3,t}$ for leaf (motion prediction vector added since apparent motion of leaf detected)</p>
<p>Final Output:</p>	<p>- $\mathbf{VSNM}'_{t,i=1.. \Theta_i}$: visual segmented (i.e., objects separated) navigation maps with motion prediction vectors if motion detected</p> <p>- \mathbf{AVNM}_t : a navigation map binding advanced auditory patterns; used in the next section</p>

	<ul style="list-style-type: none"> - $\mathbf{LNM}'_{(1, \gamma, t)}$: updated best-matching visual sensory Local Navigation Map $\mathbf{LNM}_{(1, \gamma, t)}$ including any motion prediction vectors from \mathbf{VNM}''_t - other sensory system \mathbf{LNM}s (represented by \mathbf{lnm}_i); passed through the module; used in the next section
Description:	<ul style="list-style-type: none"> -This module attempts to segment the input sensory by objects in it. -Segmentation is largely visual at present (although some limited auditory motion at present). -After segmenting a sensory scene into objects, the individual objects are sent to the Sequential/Error Correcting Module to detect motion of the objects, and if motion exists then a motion prediction vector is added to the \mathbf{VSNM} navigation maps representing each object producing \mathbf{VSNM}'. -Applies to CCA5, CCA6 and CCA7 versions of the architecture

Table A11. Summary of the Operation of the Object Segmentation Gateway Module per Equations (56) – (62)

A.6 The Causal Memory Module

This section of Appendix A applies to the CCA5 (shown in Figure 1), the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

The input sensory inputs at this point have been transformed as follows:

- visual sensory inputs $\rightarrow \mathbf{VSNM}'_{(t,i=1...\Theta_i)}$: visual segmented (i.e., objects separated) navigation maps with motion prediction vectors if motion detected
- visual sensory inputs $\rightarrow \mathbf{LNM}'_{(1, \gamma, t)}$: updated best-matching visual sensory Local Navigation Map $\mathbf{LNM}_{(1, \gamma, t)}$ including any motion prediction vectors from \mathbf{VNM}''_t
- auditory sensory inputs $\rightarrow \mathbf{AVNM}_t$: a navigation map binding auditory patterns
- olfactory sensory inputs $\rightarrow \mathbf{LNM}_{(3, \gamma, t)}$: updated best-matching olfactory sensory Local Navigation Map $\mathbf{LNM}_{(3, \gamma, t)}$
- other sensory systems inputs $\rightarrow \mathbf{LNM}_{(4...n_{\sigma}, \gamma, t)}$: updated best-matching other sensory (e.g., tactile, radar, etc.) Local Navigation Maps $\mathbf{LNM}_{(4...n_{\sigma}, \gamma, t)}$

These single sensory system navigation maps will now be matched against the previously stored multi-sensory navigation maps stored in the Causal Memory Module. The matching algorithm `Object_Seg_Mod.match_best_map` (63) chooses previously stored Navigation Map A as the best match. Thus, Navigation Map A (i.e., $\chi = [[\text{modcode}=\text{Causal_Mem_Mod}, \text{mapno}=3579], x, y, z]$) is set to become the Working Navigation Map \mathbf{WNM} (63).

\mathbf{WNM} must be updated with the actual sensory inputs, since after all it is at this point simply a retrieved best matching navigation map. If there are too many changes between the actual sensory inputs \mathbf{actual}_t (64) and an arbitrary threshold h' (65), then rather than update \mathbf{WNM} a new Working Navigation Map \mathbf{WNM}' will be created from the actual sensory inputs \mathbf{actual}_t via copying the actual sensory inputs \mathbf{actual}_t to an empty navigation map $\mathbf{TempMap}$ (69). Note from (66) that $\mathbf{TempMap}$ is actually defined as the temporary memory area “TempMap” within the Navigation Module A (Figure 5). Often values from the Causal Memory Module (e.g., a linked navigation map) or Navigation Module may have automatically been sent (i.e., copied) to $\mathbf{TempMap}$. Thus, the pseudocode `Nav_ModA.TempMap.erase()` (68) is called to ensure $\mathbf{TempMap}$ has no contents and is equivalent to an empty navigation map before using it in (69).

However, if **WNM** (i.e., which is a retrieved navigation map from the Causal Memory Module) (63) is close enough to the actual input sensory inputs **actual_t**, then the information from **actual_t** is copied to **WNM**, and creates the updated Working Navigation Map **WNM'_t** (67).

In (70) **WNM'** is stored in the Causal Memory Module—in future operations of the architecture it will also be matched against various sensory inputs. This pseudocode also specifies that there is updating of the **linkaddresses** to and from other navigation maps in the Causal Memory Module and the updated Working Navigation Map **WNM'_t**.

These equations remain largely unchanged in the CCA6 architecture from the prior CCA5 architecture, other than distinguishing the two Navigation Modules A and B. This still largely applies to the CCA7 architecture.

$$\mathbf{WNM}_t = \text{Object_Seg_Mod.match_best_map}(\mathbf{VSNM}'_{(t,i=1\dots\Theta_i)}, \mathbf{LNM}'_{(1,\gamma,t)}, \mathbf{AVNM}_t, \mathbf{LNM}_{(3,\gamma,t)}, \mathbf{LNM}_{(4\dots n_{\sigma},\gamma,t)}) \quad (63)$$

$$\mathbf{actual}_t = [\mathbf{VSNM}'_{(t,i=1\dots\Theta_i)}, \mathbf{LNM}'_{(1,\gamma,t)}, \mathbf{AVNM}_t, \mathbf{LNM}_{(3,\gamma,t)}, \mathbf{LNM}_{(4\dots n_{\sigma},\gamma,t)}] \quad (64)$$

$$h' = \text{number of differences allowed to be copied onto existing navigation map} \in \mathbb{R} \quad (65)$$

$$\mathbf{TempMap} = \text{Nav_ModA.TempMap} \in \mathbb{R}^{m \times n \times o \times p} \quad (66)$$

$$| \text{Object_Seg_Mod.differences}(\mathbf{actual}_t, \mathbf{WNM}_t) | \leq h', \\ \Rightarrow \mathbf{WNM}'_t = \mathbf{WNM}_t \cup \mathbf{actual}_t \quad (67)$$

$$\text{Nav_ModA.TempMap.erase}() \quad (68)$$

$$| \text{Object_Seg_Mod.differences}(\mathbf{actual}_t, \mathbf{WNM}_t) | > h', \\ \Rightarrow \mathbf{WNM}'_t = \mathbf{TempMap} \cup \mathbf{actual}_t \quad (69)$$

$$\text{Causal_Mem_Mod.store_WNM_update_links}(\mathbf{WNM}'_t) \quad (70)$$

Object_Seg_Mod.match_best_map (-pseudocode for an algorithm that takes processed single sensory navigation maps and matches them to the best matching multisensory navigation map in the Causal Memory Module
VSNM' _(t,i=1...Θ_i) ,	-note that this pseudocode is run from the Object Segmentation Gateway Module since all of the arguments of this pseudocode are in the latter module
LNM' _(1,γ,t) ,	
AVNM _t ,	-a small example of this algorithm is illustrated in Figure 5 where the best-matching visual local navigation map LNM' _(1,γ,t) , the visual segmentation navigation maps VSNM' _(t,i=1...Θ_i) , the navigation map binding auditory patterns AVNM _t , and the olfactory local navigation map LNM _(3,γ,t) , are matched against the multi-sensory navigation maps of the Causal Memory Module
LNM _(3,γ,t) ,	
LNM _(4...n_σ,γ,t))	-the arguments to the algorithm (VSNM' _(t,i=1...Θ_i) , LNM' _(1,γ,t) , AVNM _t , LNM _(3,γ,t) , LNM _(4...n_σ,γ,t)) are described individually below

→ WNM	-the best matching multisensory map is designated as the Working Navigation Map WNM
<p>argument used in <code>match_best_</code></p> <p>map: VSNM'_{<i>t,i=1...Θ_i</i>}</p> <p>VSNM_{<i>t,i=1...Θ_i</i>} → VSNM'_{<i>t,i=1...Θ_i</i>}</p> <p>(Note: VSNM is first created in the Object Segmentation Gateway Module, and then updated to VSNM' in the Sequential/Error Correcting Module)</p>	<p>-a “Visual Segmentation Navigation Map” VSNM is an ordinary navigation map with visually segmented information stored on it</p> <p>-the Object Segmentation Gateway Module will “segment” a sensory scene into objects of interest</p> <p>-each segmented object is treated as a separate navigation map (there is also a navigation map of all the objects together on the navigation map)</p> <p>-the subscript <i>i</i> refers to multiple VSNM’s that can be created for a given sensory scene</p> <p>-<i>i=1...Θ_i</i> refers to the full set for VSNM’s created for a given sensory scene, i.e., VSNM₁ to the last VSNM_{Θ_i} created</p> <p>-the subscript <i>t</i> refers to time since the values change each cognitive cycle</p> <p>-it is useful to calculate motion prediction vectors, if they exist (often they will not) for each segmented object (e.g., if there is a river object, a rock object and leaf floating in the river object, then it is very useful to segment the sensory scene into these objects (i.e., river, rock and leaf) and a motion prediction vector to show the motion of the leaf is very useful – thus there will be VSNM_{1,<i>t</i>} for river, VSNM_{2,<i>t</i>} for rock and VSNM_{3,<i>t</i>} for leaf, in this example)</p> <p>-continuing this example: VSNM_{1,<i>t</i>} for river, VSNM_{2,<i>t</i>} for rock and VSNM_{3,<i>t</i>} for leaf will be propagated to the Sequential/Error Correcting Module for computation of motion prediction vectors for each of these VSNM’s</p> <p>-continuing this example: a motion prediction vector is computed and stored on VSNM_{3,<i>t</i>} for leaf, but VSNM_{1,<i>t</i>} for river, VSNM_{2,<i>t</i>} for rock did not have enough motion for such vectors and are unchanged</p> <p>-continuing this example: VSNM'_{1,<i>t</i>} for river (unchanged), VSNM'_{2,<i>t</i>} for rock (unchanged) and VSNM'_{3,<i>t</i>} for leaf (motion prediction vector added) are then returned back to the Object Segmentation Gateway Module</p>
<p>argument used in <code>match_best_</code></p> <p>map:</p> <p>LNM'_(1, <i>γ_t</i>)</p>	<p>-LNM'_(1, <i>γ_t</i>) is the updated best-matching visual sensory Local Navigation Map -- LNM_(1, <i>γ_t</i>) is further updated with any visual or auditory motion prediction vectors from VNM'_{<i>t</i>} to create LNM'_(1, <i>γ_t</i>)</p>
<p>argument used in <code>match_best_</code></p> <p>map:</p> <p>AVNM_{<i>t</i>}</p>	<p>-pseudocode for an algorithm that matches <i>auditory_series_t</i> with auditory time series stored in the Sequential/Error Correcting Module</p> <p>-a best match is chosen (or if no matches close enough then <i>auditory_series_t</i> used itself as the best match)</p> <p>-the best match is then updated from <i>auditory_series_t</i></p>

Sequential_Mod.auditory_ match_process (<i>auditory_series_t</i>) → → AVNM_t	-the updated best match is stored in the Sequential/Error Correcting Module for future matching -the updated best match is then transformed into a complex advanced motion prediction vector (i.e., showing in more detail the pattern of the auditory sensations) and is output as AVNM_t (which as described above is an “Audio Vector Navigation Map”)
argument used in match_best_ map: LNM_(3, γ, t)	the local navigation map LNM stored in the Input Sensory Vectors Association Module σ with map number γ which best matches in the incoming σ sensory inputs (e.g., if $\sigma = 3$ then it would be olfactory sensory inputs)
argument used in match_best_ map: LNM_(4...n_σ, γ, t)	-for illustrative purposes only three sensory systems: visual, auditory and olfactory are often used here, but there can be additional ones such as tactile, or synthetic ones such as radar, etc. -4...n _σ means from LNM₄ to the last sensory system being used and its local navigation map LNM_{n_σ}
<i>actual_t</i>	-this is a vector that holds all the arguments for the pseudocode match_best_map = [VSNM' _(t,i=1...Θ_i) , LNM' _(1, γ, t) , AVNM_t , LNM_(3, γ, t) , LNM_(4...n_σ, γ, t)]
TempMap (NavMod A module)	-a navigation map of the same dimensions as the other navigation maps in the architecture but it is a short-term memory region in the Navigation Module A Nav_ModA.TempMap rather than being an ordinary navigation map
Object_Seg_Mod. differences()	-pseudocode for an algorithm that calculates the differences between two navigation maps -similar to Input_Assocn_Mod _σ .differences described above (21, 22) -used in (67) and (68) to calculate the differences between the processed input sensory signal <i>actual_t</i> and the retrieved best matching multi-sensory navigation map WNM_t
WNM_t ∪ actual_t → WNM'_t	- WNM must be updated with the actual sensory inputs -the processed sensory inputs <i>actual_t</i> are copied to WNM , thereby forming WNM' the updated Working Navigation Map which will be used by the Navigation Module in the section below
h' TempMap_t ∪ actual_t → WNM'_t	- WNM must be updated with the actual sensory inputs -if there are too many changes between the actual sensory inputs <i>actual_t</i> (64) and this arbitrary threshold h', then rather than update WNM , the actual sensory inputs <i>actual_t</i> will be used to create a new navigation map which is considered the

	<p>Working Navigation Map WNM' which will be used by the Navigation Module in the section below</p> <p>-as noted above TempMap is a temporary memory region in the Navigation Module A that simulates a navigation map; actual_t is being copied to an empty navigation map and results in WNM'_t</p>
WNM'_t	<p>-updated Working Navigation Map</p> <p>-the current Working Navigation Map WNM'_t is the navigation map which the Navigation Module A focuses its attention on, i.e., applies operations on to make a decision to take some sort or no action</p> <p>-as will be seen in the following sections, a navigation map can be assigned as being the WNM' for that cognitive cycle, and then a different one in the next cognitive cycle, and so on, depending on the sensory information being processed and the results obtained</p>
Causal_Mem_Mod. store_WNM_update_links (WNM'_t)	<p>-pseudocode specifying that the updated Working Navigation Map WNM'_t is stored in the Causal Memory Module</p> <p>- pseudocode specifying updating of the linkaddresses to and from the updated Working Navigation Map WNM'_t is stored in the Causal Memory Module</p> <p>-in future cognitive cycles the processed sensory inputs actual_t will be matched against the navigation maps in the Causal Memory Module which includes this newly stored navigation map</p>
Nav_ModA.TempMap.erase()	<p>-pseudocode specifying that TempMap is erased</p> <p>-thus, it will act as an empty navigation map the next time it is required</p>

Table A12. Explanation of Symbols and Pseudocode in Equations (63) – (70)

Input:	<p>-VSNM'_(t,i=1...Θ_j) : visual segmented (i.e., objects separated) navigation maps with motion prediction vectors if motion detected</p> <p>-LNM'_(1,γ,t) : best-matching visual sensory Local Navigation Map LNM_(1,γ,t) including any motion prediction vectors from VNM''_t</p> <p>-AVNM_t : a navigation map binding auditory patterns</p> <p>-LNM_(3,γ,t) : the olfactory local navigation map</p> <p>-LNM_(4...n_σ,γ,t) : other sensory local navigation maps</p> <p>(these processed sensory inputs are represented by the vector actual_t)</p>
Output:	<p>WNM'_t : updated Working Navigation Map which will be used in the Navigation Module A (as well as other modules of the architecture)</p>

Description:	<ul style="list-style-type: none"> -In this module the processed single sensory system navigation maps (<i>actual_t</i>) will be matched against the previously stored multi-sensory navigation maps stored in the Causal Memory Module. -The best matching navigation map is then updated with the actual sensory inputs, i.e., <i>actual_t</i>, and acts as the Working Navigation Map WNM' for this cognitive cycle. -The current Working Navigation Map WNM'_t is the navigation map which the Navigation Module A focuses its attention on, i.e., applies operations on to make a decision to take some sort or no action. -Applies to CCA5, CCA6 and CCA7 versions of the architecture
---------------------	--

Table A13. Summary of the Operation of the Causal Memory Module per Equations (63) – (70)

A.7 Navigation Module A

This section of Appendix A applies to the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

As seen in Figure 5 there are now in the CCA6 two Navigation Modules—Navigation Module A and Navigation Module B (as opposed to the single Navigation Module in the prior CCA5 architecture, as seen in Figure 1). As seen in Figure 9 there are now in the CCA7 multiple Navigation Module B's. However, this section still applies to the single Navigation Module A in the CCA7 architecture.

In the Navigation Module A an instinctive primitive or a learned primitive (which themselves are navigation maps) in conjunction with the Working Navigation Map **WNM'** may result in an action signal. This signal goes to the Output Vector Association Module A and to the external embodiment, completing the cognitive cycle.

In some cognitive cycles there will be no output but intermediate results from the Navigation Module A are fed back and re-operated on in the next cognitive cycle. This will be discussed in the following section. In this section, the more straightforward case is considered where a learned or instinctive primitive is applied against a Working Navigation Map **WNM'** and an action is produced.

From Figure 5 the Autonomic Module and the Goal/Emotion Module can be seen. These modules, which operate in the areas much as their names indicate, are essential for an agent such as the Causal Cognitive Architecture 6. However, these modules will not be formalized so as to focus on the Navigation Module A in this section. The *autonomic* variable (71) which reflects the energy levels and maintenance issues with embodiments of the architecture, will affect the value of the *emotion* variable (72) and the **GOAL** navigation map (73) as shown in equation (74). The **GOAL** variable reflects both intuitive and learned goals of the architecture acting as an agent and is represented as a full navigation map. The *emotion* variable will affect which features the architecture pays more attention to in their processing as well as in selecting the appropriate instinctive or learned primitive to act on the current Working Navigation Map **WNM'**.

As mentioned earlier, instinctive and learned primitives, which act as small rules or productions, are stored in modified navigation maps, called respectively instinctive primitive navigation maps **IPM**, and learned primitive navigation maps **LPM**. These primitive navigation maps have cells that contain procedures. However, the cells can also contain features and linkaddresses.

In the Navigation Module A, a learned or instinctive primitive navigation map is compared against the Working Navigation Map **WNM'**. By simple array operations and logical operations, the

Navigation Module A may produce an *action* signal, i.e., a signal to move some actuator or send an electronic signal. The *action* signal goes to the Output Vector Association Module A and then to the external embodiment.

Instinctive primitives **IPM**'s are stored in the Instinctive Primitives Module and come preprogrammed with the architecture. Learned primitives **LPM**'s in the CCA6 are now distributed among the many Input Sensory Vectors Association Modules and the Causal Memory Module (Figure 5). Both instinctive and learned primitive navigation maps have cells that contain procedures. However, the cells can also contain features and linkaddresses.

The pseudocode indicating the algorithm

`Instinctive_Primitives_Mod.match_best_primitive (actualt, emotiont, GOALt)` (76) will choose the most appropriate instinctive primitive **WIP_t** (75) to apply against the Working Navigation Map **WNM'** in the Navigation Module A. Similarly, the pseudocode indicating the algorithm `Learned_Primitives_Mod.match_best_primitive (actualt, emotiont, GOALt)` (78) will choose the most appropriate learned primitive **WLP_t** (77) to apply against the Working Navigation Map **WNM'** in the Navigation Module A.

Then a decision is made to use either the best instinctive primitive **WIP_t** or the best learned primitive **WLP_t** as the Working Primitive **WPR_t** (79) which will actually be the primitive applied against the Working Navigation Map **WNM'**. At present a simple scheme is used to make this decision. If there is no learned primitive **WLP_t** then the best instinctive primitive **WIP_t** is assigned to be the Working Primitive **WPR_t** (80). Otherwise, if a best learned primitive exists, then it is assigned to be the Working Primitive **WPR_t** (81).

At this point in the Navigation Module A there is a Working Primitive **WPR_t** and a Working Navigation Map **WNM'**. By simple array operations and logical operations, the Navigation Module A compares and operates on the Working Primitive **WPR_t** and the Working Navigation Map **WNM'**. Equation (82) shows this is indicated by pseudocode `Nav_ModA.apply_primitive(WPRt, WNM')`. As a result, an *action_t* value may be produced, which is a signal to move some actuator or send an electronic signal. The *action* signal goes to the Output Vector Association Module A and then to the external embodiment (Figure 5).

The *action* signal from the Navigation Module A may be, for example, **<move right>**. This signal is processed by the Output Vector Association Module A (Figure 5). The pseudocode `Output_Vector_Mod.action_to_output(actiont, WNM')` (84) results in **output_vector_t** which contains more detailed instructions for the actuators to cause the embodiment to move right. However, **output_vector_t** is first sent to the Sequential/Error Correcting Module where it will be corrected for motion and timing issues, resulting in a **motion_correction_t** vector (86). The pseudocode `Output_Vector_Mod.apply_motion_correction(output_vectort, motion_correctiont)` (87) produces the final signal **output_vector'**_t which is sent to the Output Vector Shaping Module (Figure 5). In the Output Vector Shaping Module the final signal shaping and low-level transformations of the signal are performed (not indicated in the equations) to directly signal movement of the embodiment's actuators or signal transmission of an electronic signal.

These equations in the CCA6 architecture are somewhat changed from the prior CCA5 architecture to reflect the duplication of the Navigation Modules. They continue to still apply to the CCA7 architecture with regard to Navigation Module A and related modules.

$$autonomic \in \mathbb{R} \quad (71)$$

$$emotion \in \mathbb{R} \quad (72)$$

$$\mathbf{GOAL} \in \mathbb{R}^{m \times n \times o \times p} \quad (73)$$

$$[emotion_t, \mathbf{GOAL}_t] = \text{Goal/Emotion_Mod.set_emotion_goal}(autonomic_t, \mathbf{WNM}'_{t-1}) \quad (74)$$

$$\mathbf{WIP} \in \mathbb{R}^{m \times n \times o \times p} \quad (75)$$

$$\mathbf{WIP}_t = \text{Instinctive_Primitives_Mod.match_best_primitive}(\mathbf{actual}_t, emotion_t, \mathbf{GOAL}_t) \quad (76)$$

$$\mathbf{WLP} \in \mathbb{R}^{m \times n \times o \times p} \quad (77)$$

$$\mathbf{WLP}_t = \text{Learned_Primitives_Mod.match_best_primitive}(\mathbf{actual}_t, emotion_t, \mathbf{GOAL}_t) \quad (78)$$

$$\mathbf{WPR} \in \mathbb{R}^{m \times n \times o \times p} \quad (79)$$

$$\mathbf{WLP}_t = [], \Rightarrow \mathbf{WPR}_t = \mathbf{WIP}_t \quad (80)$$

$$\mathbf{WLP}_t \neq [], \Rightarrow \mathbf{WPR}_t = \mathbf{WLP}_t \quad (81)$$

$$action_t = \text{Nav_ModA.apply_primitive}(\mathbf{WPR}_t, \mathbf{WNM}'_t) \quad (82)$$

$$\mathbf{output_vector} \in \mathbb{R}^n \quad (83)$$

$$action_t = [\text{"move*"}], \Rightarrow \mathbf{output_vector}_t = \text{Output_Vector_Mod.action_to_output}(action_t, \mathbf{WNM}'_t) \quad (84)$$

$$\mathbf{motion_correction} \in \mathbb{R}^2 \quad (85)$$

$$action_t = [\text{"move*"}], \Rightarrow \mathbf{motion_correction}_t = \text{Sequential_Mod.motion_correction}(action_t, \mathbf{WNM}'_t, \mathbf{visual_series}_t) \quad (86)$$

$$\mathbf{output_vector}'_t = \text{Output_Vector_Mod.apply_motion_correction}(\mathbf{output_vector}_t, \mathbf{motion_correction}_t) \quad (87)$$

<i>autonomic</i>	variable which reflects the energy levels and maintenance issues with embodiments of the architecture
<i>emotion</i>	variable which affects which features the architecture pays more attention to
GOAL	reflects both intuitive and learned goals of the architecture acting as an agent, and is represented as a full navigation map
Goal/Emotion_Mod.set_emotion_goal(<i>autonomic</i> _t , WNM' _{t-1})	-pseudocode that calculates the values of the <i>emotion</i> and GOAL variables

→ [<i>emotion_t</i> , GOAL_t]	<p>-Goal/Emotion_Mod refers to the Goal/Emotion Module</p> <p>-the argument WNM'_{t-1} is the Working Navigation Map from the previous (i.e., t-1) cognitive cycle</p>
Instinctive_Primitives_Mod. match_best_primitive (<i>actual_t</i> , <i>emotion_t</i> , GOAL_t) → WIP_t	<p>-pseudocode that returns the most appropriate instinctive primitive navigation map IPM from the Instinctive Primitives Module given the arguments of the processed sensory inputs <i>actual_t</i>, the current <i>emotion_t</i> value, and the current GOAL_t navigation map</p> <p>-Instinctive_Primitives_Mod refers to the Instinctive Primitives Module</p> <p>-the most appropriate instinctive primitive IPM returned is considered to be the Working Instinctive Primitive WIP_t</p>
Learned_Primitives_Mod.match_ best_primitive (<i>actual_t</i> , <i>emotion_t</i> , GOAL_t) → WLP_t	<p>-pseudocode that returns the most appropriate learned primitive navigation map LPM from the Instinctive Primitives Module given the arguments of the processed sensory inputs <i>actual_t</i>, the current <i>emotion_t</i> value, and the current GOAL_t navigation map</p> <p>-Learned_Primitives_Mod refers to the Learned Primitives Module</p> <p>-the most appropriate instinctive primitive LPM returned is considered to be the Working Learned Primitive WLP_t</p>
WLP_t = [], ⇒ WPR_t = WIP_t WLP_t ≠ [], ⇒ WPR_t = WLP_t → WPR	<p>-the Working Primitive WPR is the primitive that will be applied against the Working Navigation Map WNM' in the Navigation Module A</p> <p>-it is composed of inputs from the Working Instinctive Primitive WIP and the Working Learned Primitive WLP</p> <p>-currently a simple scheme to calculate it is used since there is only a very small number of learned primitives LPM's—if a Working Learned Primitive WLP exists then WLP_t will be used as WPR_t, otherwise the Working Instinctive Primitive WIP_t will be used as WPR_t</p>
<i>procedure</i>	<p>-a cell in a navigation map can hold values which are <i>procedure</i> variables (35), <i>feature</i> variables (34), and linkaddress <i>χ'</i> variables (36)</p> <p>-a <i>procedure</i> value could be, for example, <move right> which would indicate that the embodiment of the architecture should move towards the right</p>
Nav_ModA.apply_primitive(WPR_t , WNM'_t) → <i>action_t</i>	<p>-Nav_ModA refers to the Navigation Module A (Figure 5)</p> <p>-apply_primitive(WPR_t, WNM'_t) is pseudocode for an algorithm which applies the Working Primitive WPR_t to the Working Navigation Map WNM', often producing an <i>action_t</i> value</p> <p>-note that in some cognitive cycles no <i>action_t</i> value may result from the current WPR_t and WNM'_t</p> <p>-in other cognitive cycles instead of an actionable <i>action_t</i> value there is the feeding back of intermediate results for re-processing in the next cognitive cycle (discussed in the next section)</p>

	<p>-the mechanism of producing an $action_t$ value is via simple array operations (both \mathbf{WPR}_t and \mathbf{WNM}'_t are arrays) and very simple logical operations that could be biologically feasible as possible</p>
<p>Output_Vector_Mod. $action_to_output(action_t, \mathbf{WNM}'_t)$ $\rightarrow output_vector_t$</p>	<p>-Output_Vector_Mod refers to the Output Vector Association Module A</p> <p>-pseudocode describing taking the $action_t$ value and the current Working Navigation Map \mathbf{WNM}'_t and creating $output_vector_t$ which would provide more detailed instructions for the actuators of the embodiment in order to carry out the $action_t$ value</p> <p>-for example, the $action$ value could be, for example, <move right> which this pseudocode would transform into $output_vector_t$ which provides more detailed instructions for the embodiment's actuators with regard to moving to the right</p>
<p>Sequential_Mod. $motion_correction(action_t, \mathbf{WNM}'_t, visual_series_t)$ $\rightarrow motion_correction_t$</p>	<p>-Sequential_Mod refers to the Sequential/Error Correcting Module</p> <p>-pseudocode which produces a motion correction value based on the relative movement of the embodiment itself and the motion which is desired to achieve</p> <p>-the pseudocode considers what $action_t$ is desired, the current Working Navigation Map \mathbf{WNM}'_t and the current movement of the embodiment via the visual movement detected via $visual_series_t$ (44, 45)</p> <p>-in the future additional sensory systems (e.g., a positioning system, an inertial system, and so on) could be used in addition to the information in the $visual_series_t$</p> <p>-the vector $motion_correction_t$ is computed and returned to the Output Vectors Association Module</p> <p>-for example, the current $action$ value is <move right> so the pseudocode would compute if any motion correction was required based on the current actual movement of the embodiment of the architecture</p>
<p>Output_Vector_Mod. $apply_motion_correction(output_vector_t, motion_correction_t)$ $\rightarrow output_vector'_t$</p>	<p>-Output_Vector_Mod refers to the Output Vector Association Module A</p> <p>-pseudocode that applies the computed vector $motion_correction_t$ against the previously computed $output_vector_t$</p> <p>-an updated $output_vector'_t$ is produced</p> <p>$\rightarrow output_vector'_t$ provides more detailed and motion corrected instructions for the actuators in this case, e.g, <move right>, to move the embodiment to the right</p>

Table A14. Explanation of Symbols and Pseudocode in Equations (71) – (87)

Input:	<p>WPR_t: the Working Primitive WPR is the primitive that will be applied against the Working Navigation Map WNM' in the Navigation Module A</p> <p>WNM'_t: the current Working Navigation Map which is derived from the processed sensory inputs</p>
Output:	<p><i>action_t</i> : a signal to move some actuator or send an electronic signal e.g., <move right></p> <p>it is sent to the Output Vector Association Module A where more detailed instructions are created to effect the required actuator outputs</p>
Description:	<p>-The Navigation Module A applies the Working Primitive WPR_t against the current Working Navigation Map WNM' and may produce an <i>action_t</i> signal (sometimes no signal is produced or sometimes there is a signal to feedback intermediate results, discussed in the next section).</p> <p>-Applies to the CCA5 version of the architecture (single Navigation Module) and to the CCA6 and CCA7 versions of the architecture with respect to Navigation Module A and associated modules</p>

Table A15. Summary of the Operation of the Navigation Module A per Equations (71) – (87)

A.8 Feedback Signals and Intermediate Results

This section of Appendix A applies to the CCA5 version of the architecture (Figure 1) but there is only a single Navigation Module. This section of the Appendix A applies CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture where there is a separate Navigation Module A and B (with the latter plural in the CCA7).

The Causal Cognitive Architecture, including the CCA6, makes use of feedback pathways—states of a downstream module can influence the recognition and processing of more upstream sensory inputs. This is advantageous in order to better recognize noisy or incomplete input sensory information. In previous versions of the architecture and again in the CCA6, the feedback pathways between the Input Sensory Vectors Association Modules and the Navigation Module A remain enhanced so that they can allow not just a feedback signal, but the full intermediate results from the Navigation Module A to be stored in the Input Sensory Vectors Association Modules.

In some cognitive cycles in the CCA6 there is no output signal. In certain states (for example, no actionable output from the Navigation Module A) the information in the Navigation Module A can be fed back and stored in the Input Sensory Vectors Association Modules. When this happens then in the next cognitive cycle these intermediate results will automatically be considered as the input sensory information and propagated to the Navigation Module A and operated on again. As noted above in the body of the paper, by feeding back and re-operating on the intermediate results, the Causal Cognitive Architecture can formulate and explore possible cause and effect of actions, i.e., generate causal behavior.

The Working Primitive **WPR_t** has been applied against the Working Navigation Map **WNM'_t** producing an *action_t* signal: *action_t* = Nav_ModA.apply_primitive(**WPR_t**, **WNM'_t**) (82). In the previous section the *action_t* signal was actionable, i.e., it contained the string “move*” and so via equations (84–87) it was propagated to the Output Vector Association Module A and to the Output Vector Shaping Module A where it then moved the actuators of the embodiment or sent an electronic signal in accordance with the *action* specified. However, what if the *action_t* signal produced is not

actionable, i.e., it does not contain the string “move*”? There is no indication that any actuator should be moved, or any electronic signal should be sent. In this case $action_t$ would be discarded. Another $action_t$ signal would be produced in the next cognitive cycle, and perhaps it would be actionable.

However, given the enhancement in feedback pathways between the Navigation Module A and the Input Sensory Vectors Association Modules, the Causal Cognitive Architecture can in these cases feed back the intermediate results of the Navigation Module A and store them in the Input Sensory Vectors Association Modules (88). Note that equation (88) will feedback the intermediate results (i.e., the Working Navigation Map \mathbf{WNM}'_t if $action_t \neq \text{“move*”}$, i.e., the $action_t$ signal is not actionable. However, if the Working Primitive \mathbf{WPR}_t explicitly specifies to discard the intermediate results they will not be fed back. Similarly, if the Working Primitive \mathbf{WPR}_t explicitly specifies to feed back the intermediate results they will be fed back even if they are actionable. In equation (88) the pseudocode `Nav_ModA.feedback_to_assocn_mod(\mathbf{WNM}'_t)` specifies to feed back and store \mathbf{WNM}'_t in the Input Sensory Vectors Association Modules.

In the next cognitive cycle, in equation (89) the pseudocode `Input_Sens_Vectors_Assoc_Module σ .extract_ σ (\mathbf{WNM}'_t)` specifies that the Working Navigation Map just fed back (i.e., \mathbf{WNM}'_t) is to have its components extracted (i.e., effectively stored) into the best matching local navigation maps of the various Input Sensory Vectors Associations Modules in the next cognitive cycle $t+1$. Thus, as the cognitive cycle progresses, the actual sensory inputs will be ignored, and instead these components of the previous Working Navigation Map \mathbf{WNM}'_t (i.e., the intermediate results from the Navigation Module A in the last cognitive cycle) are propagated towards the Navigation Module A and operated on again, perhaps by new instinctive or learned primitives in this new cognitive cycle. Feeding back the Navigation Module A’s intermediate results back to the sensory stages and then processing the intermediate results in the next cycle by the Navigation Module A, over and over again as needed, will allow a robust application of rules to be applied onto the modeled world, and allow exploration and examination of what the effect of a cause will be. As described in the text of the paper above, previous Causal Cognitive Architectures have shown that by feeding back and re-operating on the intermediate results, the architecture can generate causal behavior.

These equations in the CCA6 architecture are somewhat changed from the prior CCA5 architecture to reflect the duplication of the Navigation Modules.

$$\begin{aligned}
 & (action_t \neq \text{“move*”} \text{ and } \mathbf{WPR}_t \neq [\text{“discard*”}]) \text{ or } \mathbf{WPR}_t = [\text{“feedback*”}], \\
 & \Rightarrow \text{Nav_ModA.feedback_to_assocn_mod}(\mathbf{WNM}'_t) \quad (88) \\
 & \Rightarrow \forall \sigma: \mathbf{LNM}_{(\sigma, \gamma, t+1)} = \text{Input_Sens_Vectors_Assoc_Module}_{\sigma}.\text{extract_}\sigma(\mathbf{WNM}'_t) \quad (89)
 \end{aligned}$$

$(action_t \neq \text{“move*”} \text{ and } \mathbf{WPR}_t \neq [\text{“discard*”}])$ or $\mathbf{WPR}_t = [\text{“feedback*”}],$	-if $action_t$ which the Navigation Module A (e.g., Figure 5 or Figure 9) produced (82) is not actionable (i.e., does not specify an activation of an actuator or the transmission of an electronic signal) and also there is no specification by the Working Primitive \mathbf{WPR}_t to discard $action_t$ (i.e., the
--	---

$\Rightarrow \dots$	<p>Navigation Module A should not do anything this cognitive cycle) then operations after the \Rightarrow symbol will occur</p> <p>-if Working Primitive WPR_t specifies that the intermediate results of the Navigation Module A should be fed back, then regardless of whether <i>action</i>_t is actionable or not, the operations after the \Rightarrow symbol will occur</p>
$\Rightarrow \text{Nav_ModA.feedback_to_assocn_mod}(\mathbf{WNM}'_t)$	-pseudocode to feed back and store the intermediate results of the Navigation Module A, i.e., the Working Navigation Map WNM' _t to the Input Sensory Vectors Association Modules
$\Rightarrow \forall \sigma: \mathbf{LNM}_{(\sigma, \gamma, t+1)} =$ $\text{Input_Sens_Vectors_Assoc_Module}.\text{extract_}\sigma(\mathbf{WNM}'_t)$	<p>-pseudocode that specifies that the Working Navigation Map just fed back (i.e., WNM'_t) is to have its components extracted (i.e., effectively stored) into the best matching local navigation maps of the various Input Sensory Vectors Associations Modules of the next cognitive cycle t+1</p> <p>-thus, as the cognitive cycle progresses, the actual sensory inputs will be ignored, and instead these components of the previous Working Navigation Map WNM'_t (i.e., the intermediate results from the Navigation Module A in the last cognitive cycle) will be propagated towards the Navigation Module A and operated on again</p>

Table A16. Explanation of Symbols and Pseudocode in Equations (88) – (89)

Initial Input:	<i>not applicable as this is a continuation of Navigation Module A operations</i>
Initial Output:	WNM' _t : the current Working Navigation Map is fed back to the Input Sensory Vectors Association Modules
Second Input:	in the next cognitive cycle LNM _($\sigma, \gamma, t+1$) for some or all of the sensory systems, derived from the fed back WNM' _t , are treated as inputs and processed again by the Navigation Module A complex (i.e., the Object Segmentation Gateway Module, the Causal Memory Module and the Navigation Module A)
Second Output:	<p>-in the next cognitive cycle, the Navigation Module A will produce again an <i>action</i>_t signal as per equation (82)</p> <p>-if <i>action</i>_t is actionable (i.e., an activation of an actuator or the transmission of an electronic signal) then the <i>action</i>_t signal will be sent to the Output Vector Association Module A, as in the previous section</p> <p>-if <i>action</i>_t is not actionable then intermediate results of the Navigation Module A (i.e., WNM'_t) may be fed back again to the Input Sensory Vectors Association Modules (88, 89)</p>
Description:	<p>-In the previous section it was seen that the Navigation Module A applies the Working Primitive WPR_t against the current Working Navigation Map WNM' and produced an <i>action</i>_t signal which was sent to the Output Vector Association Module A and then to the external embodiment to activate an actuator or send an electronic signal.</p> <p>-In this section it is seen that if <i>action</i>_t is not actionable then intermediate results of the Navigation Module A (i.e., WNM'_t) may be fed back again to the Input Sensory Vectors Association Modules and then in the next cognitive cycle return back to the Navigation Module A where it can be operated on again.</p>

	<p>-By feeding back and re-operating on the intermediate results, the architecture can generate causal behavior.</p> <p>-Applies to the CCA5 version of the architecture (Figure 1) but there is only a single Navigation Module. Applies CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture where there is a separate Navigation Module A and B (with the latter plural in the CCA7).</p>
--	---

Table A17. Summary of the Feedback Operations from the Navigation Module A per Equations (88) – (89)

A.9 Analogical Problem Solving

This section of Appendix A applies to the CCA5 (Figure 1), CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture. In the CCA5 version of the architecture its sole Navigation Module should be considered the same as Navigation Module A below.

In the last section above, it was shown how with enhanced feedback pathways from the Navigation Module A to the Input Sensory Vectors Association Modules and some small changes, if there is no actionable output from the Navigation Module A, then the Navigation Module A can feed back the Working Navigation Map **WNM'**. Without many changes to the architecture, when this happens then in the next cognitive cycle these intermediate results will automatically be considered as the input sensory information and propagated back to the Navigation Module A and operated on again. As discussed in the text of the paper above, it has previously been shown that in this type of Causal Cognitive Architecture that by feeding back and re-operating on the intermediate results, the architecture is sometimes able to formulate and explore possible cause and effect of actions, i.e., generate causal behavior.

However, often the combinations of sensory inputs leading to **WNM'** and the chosen instinctive or learned primitives leading to **WPR'** which operates on the Working Navigation Map **WNM'**, does not give a causally related or even a useful output.

This section describes a small, evolutionarily plausible modification of the feedback algorithm from the Navigation Module A to the Input Sensory Vectors Association Modules which readily emerges. The result is the ready generation of analogical results that may be more useful than simply feeding back and returning the intermediate results unchanged in the next cognitive cycle.

In the last section above it was seen in (88) that if the $action_t$ which is produced by the Navigation Module A (i.e., $action_t = \text{Nav_ModA.apply_primitive}(\mathbf{WPR}_t, \mathbf{WNM}'_t)$ (82)) is not actionable (i.e., there is no actuator to move or no electronic signal to send) then the Navigation Module A feeds back what will now be considered intermediate results of the Navigation Module A (i.e., \mathbf{WNM}'_t) for storage in the Input Sensory Vectors Association Modules. (A property which can easily emerge by simply enhancing feedback pathways in the architecture.) In the next cognitive cycle, \mathbf{WNM}'_t is treated as the input signal (89) and thus automatically propagated back to the Navigation Module A where it can be operated on. Perhaps different instinctive or learned primitives operating on \mathbf{WNM}'_{t-1} (“ $t - 1$ ” just means it is from the previous cognitive cycle) or the transformation of the data, will produce a useful, actionable output during this cognitive cycle. Equations (88) and (89) are reproduced again below. Note that the Working Primitive \mathbf{WPR}_t can specify to discard results (i.e., $\mathbf{WPR}_t = [\text{“discard*”}]$) after a cognitive cycle (i.e., nothing will happen except wait for the next

cognitive cycle) or it can specify to feed back the results of the Navigation Module A even if the action is actionable (i.e., $\mathbf{WPR}_t = [\text{"feedback*"}]$).

$$\begin{aligned} & (action_t \neq \text{"move*"} \text{ and } \mathbf{WPR}_t \neq [\text{"discard*"}]) \text{ or } \mathbf{WPR}_t = [\text{"feedback*"}], \\ & \Rightarrow \text{Nav_ModA.feedback_to_assocn_mod}(\mathbf{WNM}'_t) \quad (88) \\ & \Rightarrow \forall \sigma: \mathbf{LNM}_{(\sigma, \gamma, t)} = \text{Input_Sens_Vectors_Assoc_Module}_{\sigma}.\text{extract}_{\sigma}(\mathbf{WNM}'_{t-1}) \quad (89) \end{aligned}$$

These small changes in the architecture are sufficient to easily allow the emergence of a different type of feedback by the Navigation Module A. The feedback shown in (88) and (89) will no longer occur by default. However, if the Working Primitive \mathbf{WPR}_t specifically signals “feedback” then the same feedback algorithm as before (i.e., described above (88, 89)) will occur. Equations (88, 89) are replaced by (88a, 89) to specify this condition where the previously described feedback algorithm will still occur:

$$\begin{aligned} & \mathbf{WPR}_t = [\text{"feedback*"}], \\ & \Rightarrow \text{Nav_ModA.feedback_to_assocn_mod}(\mathbf{WNM}'_t) \quad (88a) \\ & \Rightarrow \forall \sigma: \mathbf{LNM}_{(\sigma, \gamma, t)} = \text{Input_Sens_Vectors_Assoc_Module}_{\sigma}.\text{extract}_{\sigma}(\mathbf{WNM}'_{t-1}) \quad (89) \end{aligned}$$

The new default feedback algorithm which will occur is now specified as starting in (90). If the $action_t$ signal produced by the Navigation Module A (i.e., $action_t = \text{Nav_ModA.apply_primitive}(\mathbf{WPR}_t, \mathbf{WNM}'_t)$ (82)) is not actionable ($action_t \neq \text{"move*"}—$ there is no actuator to move or no electronic signal to send) then fed back are what will now be considered intermediate results of the Navigation Module A (i.e., \mathbf{WNM}'_t) for storage in the Input Sensory Vectors Association Modules (90). This is the same effect as occurred before and still occurs in (88a). (Note also from (90) that if the Working Primitive \mathbf{WPR}_t specifies “analogical” feedback then this will occur even if the $action_t$ signal is actionable. Similarly, if \mathbf{WPR}_t specifies to “discard” the results, then nothing will happen except wait for the next cognitive cycle.) Thus, \mathbf{WNM}'_t is sent (i.e., fed back) to the Input Sensory Vectors Association Modules as before (90).

\mathbf{WNM}'_t is also sent to the Causal Memory Module (91). Sending \mathbf{WNM}'_t to the Causal Memory Module (91) triggers $\text{Causal_Mem_Mod.match_best_map}(\mathbf{WNM}'_t)$. This pseudocode matches \mathbf{WNM}'_t against the multisensory navigation maps stored in the Causal Memory Module, and assigns this best matching navigation map as the new \mathbf{WNM}'_t value. This matching algorithm is similar to other matching algorithms for navigation maps used by the architecture. \mathbf{WNM}'_t is also sent to the TempMap region of the Navigation Module A. This happens automatically.

Results of almost all operations from the Navigation Module A and the Causal Memory Module are routinely copied to the TempMap region. Often the information in the TempMap region (i.e., “TempMapA”, the TempMap region in Navigation Module A) is not used and it is overwritten in the next cognitive cycle. This is the case now—the contents of the TempMap region will not be used. However, in later steps in equations (92, 93) the TempMap region is indeed used. Note that in the equations the TempMap region is defined as an array **TempMap** which acts like an ordinary navigation map (66).

In (92) the pseudocode $\text{Nav_ModA.use_linkaddress1_map}(\mathbf{WNM}'_t)$ activates in the Causal Memory Module the navigation map which the most recently used linkaddress used in the past by the current Working Navigation Map \mathbf{WNM}'_t points to. As shown in (92) this result of the Causal

Memory Module is automatically copied to **TempMap_t**. (As mentioned above results of the Navigation Module A and the Causal Memory Module are routinely and automatically copied to **TempMap**.)

(Note that other algorithms besides `use_linkaddress1_map()` are possible. For example, rather than the most recently used linkaddress, a number of the linkaddresses used by this navigation map in the past can be explored and one particular linkaddress chosen, and so on. Similarly, most recently produced *action*'s can also be explored.)

In (93) the pseudocode `Nav_ModA.subtract(TempMap, WNM't)` the navigation map **WNM'_t** is subtracted from **TempMap**. (The navigation maps are arrays and can be treated as such via simple array operations.) The result becomes the new Working Navigation Map **WNM'_t** (93).

The next cognitive cycle then occurs. At this point, the contents of the **WNM'_t** Working Navigation Map in the Navigation Module A contains the difference between the previous **WNM'_t** and retrieved *linkaddress* χ ' Navigation Map stored in **TempMap**. At this point, the Input Sensory Vectors Association Modules contain the original Working Navigation Map **WNM'_{t-1}** (*t-1* just means something from the previous cognitive cycle) which was fed back and stored here. Thus, during this cognitive cycle, the actual sensory inputs will be ignored, and the previous **WNM'_{t-1}** will be propagated towards the Navigation Module A.

To indicate continuation of the operations on associated equations (90–93), in (94) it is specified “(*action_{t-1}* ≠ “move*” or **WPR_{t-1}** = [“analogical*”]) and **WPR_{t-1}** ≠ [“discard*”] and **WPR_{t-1}** ≠ [“feedback*”]),” and then specified to run this specific pseudocode: **WNM'_t** = `Nav_ModA.retrieve_and_add_vector_assocn()` (94). Rather than propagate **WNM'_{t-1}** towards the Navigation Module A and then it replace the existing **WNM'_t** and is operated on by the current Working Primitive **WPR'**, as occurred automatically with the previous feedback algorithm above, the pseudocode `retrieve_and_add_vector_assocn()` specifies that **WNM'_{t-1}** should not replace but simply be added to the existing **WNM'_t**.

Now the original Working Navigation Map **WNM'_t** in the Navigation Module A also contains the action that occurred in the past of a similar Working Navigation Map in a possible analogical situation.

The equations below are somewhat changed from the prior CCA5 architecture to reflect the duplication of the Navigation Modules. As noted above, while applying more now the CCA6 and CCA7 versions of the architecture, for the CCA5 version its sole navigation module should be considered as Navigation Module A.

((*action_t* ≠ “move*” or **WPR_t** = [“analogical*”]) and **WPR_t** ≠ [“discard*”] and **WPR_t** ≠ [“feedback*”]),
 \Rightarrow `Nav_ModA.feedback_to_assocn_mod(WNM't)` (90)
 \Rightarrow **WNM'_t** = `Causal_Mem_Mod.match_best_map(WNM't)` (91)
 \Rightarrow **TempMap_t** = `Nav_ModA.use_linkaddress1_map(WNM't)` (92)
 \Rightarrow **WNM'_t** = `Nav_ModA.subtract(WNM't, TempMap)` (93)

((*action_{t-1}* ≠ “move*” or **WPR_{t-1}** = [“analogical*”]) and **WPR_{t-1}** ≠ [“discard*”] and **WPR_{t-1}** ≠ [“feedback*”]),
 \Rightarrow **WNM'_t** = `Nav_ModA.retrieve_and_add_vector_assocn()` (94)

$WPR_t = ["feedback*"],$ $\Rightarrow \dots$	<p>-replaces equation (88) (see Table A17)</p> <p>-previously described feedback algorithm now only occurs if the Working Primitive WPR_t specifies it (i.e., $WPR_t = ["feedback*"],$).</p>
$\Rightarrow Nav_ModA.feedback_to_assoc_n_mod(WNM'_t)$	<p>-previously described in Table A17</p> <p>-applied to equations (88)/(88a) or (90)</p> <p>-pseudocode to feed back and store the intermediate results of the Navigation Module A, i.e., the Working Navigation Map WNM'_t in the Input Sensory Vectors Association Modules (Figure 5 or Figure 9)</p>
$\Rightarrow \forall \sigma: LNM_{(\sigma, \gamma, t)} =$ $Input_Sens_Vectors_Assoc_Module_{\sigma}.extract_{\sigma}(WNM'_{t-1})$	<p>-previously described in Table A17</p> <p>-applies to equation (89)</p> <p>-pseudocode that specifies that the Working Navigation Map just fed back (i.e., WNM'_{t-1} which is WNM'_t of the previous cognitive cycle) is to have its components extracted (i.e., effectively stored) into the best matching local navigation maps of the various Input Sensory Vectors Associations Modules</p> <p>-thus, as the cognitive cycle progresses, the actual sensory inputs will be ignored, and instead these components of the previous Working Navigation Map WNM'_{t-1} (i.e., the intermediate results from the Navigation Module A in the last cognitive cycle) will be propagated towards the Navigation Module A and operated on again</p> <p>-see Figure 5 (CCA6 version) or Figure 9 (CCA7 version of the architecture) for overview of modules and interconnections</p>
$(action_t \neq "move*" \text{ or } WPR_t = ["analogical*"]) \text{ and } WPR_t \neq ["discard*"] \text{ and } WPR_t \neq ["feedback*"]$ \Rightarrow	<p>-if the $action_t$ signal produced by the Navigation Module A (i.e., $action_t = Nav_ModA.apply_primitive(WPR_t, WNM'_t)$ (82)) is not actionable ($action_t \neq "move*"$—there is no actuator to move or no electronic signal to send)</p> <p>-or if the Working Primitive WPR_t specifies “analogical” feedback (regardless if the $action_t$ signal is actionable)</p> <p>-and the Working Primitive WPR_t does not specify to “discard” the results (regardless if the $action_t$ signal is actionable, if “discard” is specified then nothing happens except to wait for the next cognitive cycle)</p> <p>-and the Working Primitive WPR_t does not specify to use the previous “feedback” pseudocode (i.e., equations (88, 89)) (regardless if the $action_t$ signal is actionable)</p> <p>-then equations/pseudocode which follows—which are equations (90,91,92, 93)—will be run</p>
$\Rightarrow Nav_ModA.feedback_to_assoc_n_mod(WNM'_t)$	<p>-equation (90) is the same pseudocode as equations (88)/(88a)</p> <p>-pseudocode to feed back and store the intermediate results of the Navigation Module A, i.e., the Working Navigation Map WNM'_t in the Input Sensory Vectors Association Modules</p>
$\Rightarrow Causal_Mem_Mod.match_best_map(WNM'_t)$ $\rightarrow WNM'_t$	<p>-WNM'_t is also sent to the Causal Memory Module (91)</p> <p>-sending WNM'_t to the Causal Memory Module (91) triggers $Causal_Mem_Mod.match_best_map(WNM'_t)$</p> <p>-this pseudocode matches WNM'_t against the multisensory navigation maps stored in the Causal Memory Module, and assigns this best matching navigation map as the new WNM'_t value</p>

$\Rightarrow \text{Nav_ModA.use_linkaddress1_map}(\mathbf{WNM}'_t)$ $\rightarrow \text{TempMap}_t$	<p>-the pseudocode $\text{Nav_ModA.use_linkaddress1_map}(\mathbf{WNM}'_t)$ activates in the Causal Memory Module the navigation map which the most recently used linkaddress used in the past by the current Working Navigation Map \mathbf{WNM}'_t points to (92)</p> <p>-the result of the Causal Memory Module is automatically copied to TempMap.</p> <p>-as noted earlier, in the CCA6 version of the architecture, results of the Navigation Module A and the Causal Memory Module are routinely and automatically copied to TempMap</p>
$\Rightarrow \text{Nav_ModA.subtract}(\mathbf{WNM}'_t, \text{TempMap})$ (93) $\rightarrow \mathbf{WNM}'_t$	<p>-the pseudocode $\text{Nav_ModA.subtract}(\text{TempMap}, \mathbf{WNM}'_t)$ subtracts the navigation map \mathbf{WNM}'_t from the navigation map in TempMap (93)</p> <p>-the result becomes the new Working Navigation Map \mathbf{WNM}'_t</p>
$((action_{t-1} \neq \text{"move*"} \text{ or } \mathbf{WPR}_{t-1} = [\text{"analogical*"}]) \text{ and } \mathbf{WPR}_{t-1} \neq [\text{"discard*"}] \text{ and } \mathbf{WPR}_{t-1} \neq [\text{"feedback*"}])$, \Rightarrow	<p>-"t-1" simply refers to the cognitive cycle before this one, i.e., to the values from the previous cognitive cycle</p> <p>-if true then the pseudocode of equation (94) is run</p> <p>-if equation (90)'s conditions are true and its pseudocode is run, then this will also be true (exact same conditions except referring to the previous cognitive cycle), thus equations (90) to (94) are run one after another</p>
$\Rightarrow \text{Nav_ModA.retrieve_and_add_vector_assocn}()$ (94) $\rightarrow \mathbf{WNM}'_t$	<p>-rather than propagate \mathbf{WNM}'_{t-1} (which is in the Sensory Input Vectors Associations Modules, and will utilized in the new cognitive cycle, rather than the actual sensory inputs) towards the Navigation Module A and then automatically replace the existing \mathbf{WNM}'_t, the pseudocode specifies that \mathbf{WNM}'_{t-1} should not replace but simply be added to the existing \mathbf{WNM}'_t</p> <p>-thus, now the original Working Navigation Map \mathbf{WNM}'_t in the Navigation Module A will contain the action or navigation map that occurred in the past of a similar Working Navigation Map in a possible analogical situation</p>

Table A18. Explanation of Symbols and Pseudocode in Equations (88a) – (94)

Input:	<i>not applicable as this is a continuation of Navigation Module A operations</i>
Temporary Inputs/Outputs occurring:	<p>note: to distinguish between the several different navigation maps which are considered the current Working Navigation Map \mathbf{WNM}' at some point, descriptive suffixes are added to \mathbf{WNM}' (however, note that there is only one navigation map in the Navigation Module A at one time designated as \mathbf{WNM}'; the descriptive suffixes are for the reader, they do not exist in the architecture)</p> <p>-$\mathbf{WNM}'_{t-original}$: the original current Working Navigation Map is fed back to the Input Sensory Vectors Association Modules (90)</p> <p>- $\mathbf{WNM}'_{t-original}$: the original current Working Navigation Map is propagated to the Causal Memory Module and the resultant best matching navigation map is sent back to the Navigation Module A and replaces $\mathbf{WNM}'_{t-original}$ with $\mathbf{WNM}'_{t-best_match}$ (91)</p> <p>- $\mathbf{WNM}'_{t-best_match-linkaddress1}$: its most recently used <i>linkaddress</i> (i.e., pointing and accessing another navigation map) is sent to the Causal Memory Module and puts the retrieved (actually just activated) navigation map into TempMap (92)</p>

	<p>-TempMap: holding the retrieved navigation map $\mathbf{WNM}'_{r_best_match_linkaddress1}$ (92), which will then subtract the current Working Navigation Map $\mathbf{WNM}'_{r_best_match}$ and the result will be the new Working Navigation Map $\mathbf{WNM}'_{r_difference}$ (93)</p> <p>-in the next cognitive cycle $\mathbf{LNM}_{(a,r,t)}$ for some or all of the sensory systems, derived from the feedback $\mathbf{WNM}'_{r_original}$, are treated as inputs and added to the existing Working Navigation Map $\mathbf{WNM}'_{r_difference}$ resulting in $\mathbf{WNM}'_{r_analogical}$ (94)</p>
Output:	<i>not applicable as this is a continuation of Navigation Module A operations</i>
Description:	<p>-Earlier it was seen that the Navigation Module A applies the Working Primitive \mathbf{WPR}_t against the current Working Navigation Map \mathbf{WNM}'_t and produced an $action_t$ signal which was sent to the Output Vector Association Module A and then to the external embodiment to activate an actuator or send an electronic signal.</p> <p>-In the previous section it was seen that if $action_t$ is not actionable then intermediate results of the Navigation Module A (i.e., \mathbf{WNM}'_t) may be fed back again to the Input Sensory Vectors Association Modules and then in the next cognitive cycle return back to the Navigation Module A where it can be operated on again. By feeding back and re-operating on the intermediate results, the architecture can generate causal behavior.</p> <p>-In this section shown is a different “analogical” feedback algorithm to use if $action_t$ is not actionable.</p> <p>-The analogical feedback algorithm in this section starts off similarly to the previous feedback algorithm, but rather than simply feeding back the Working Navigation Map \mathbf{WNM}'_t unchanged to be processed further in the next cognitive cycle, it feeds back and constructs a navigation map that occurred in the past of a similar Working Navigation Map \mathbf{WNM}'_t (which can call “$\mathbf{WNM}'_{r_analogical}$” to distinguish it from the other navigation maps that are set as \mathbf{WNM}'_t at different points).</p> <p>-$\mathbf{WNM}'_{r_analogical}$ may be more likely in a possible analogical situation. This navigation map can then be processed further in the next cognitive cycle.</p> <p>-By using a possibly analogical navigation map it makes it more likely the navigation map which becomes the Working Navigation Map \mathbf{WNM}_t in the next cognitive cycle, and upon which possibly another (or possibly the same) \mathbf{WPR}_t will be applied, that an actionable $action_t$ will result since it is using a navigation map that was tried in the past.</p> <p>-At the present, the algorithm underlying the pseudocode <code>Nav_ModA.use_linkaddress1_map (\mathbf{WNM}'_t)</code> (92) is simple and straightforward. However, in future versions more sophisticated algorithms for this pseudocode could choose better among matching navigation maps that were more successful in the past in producing an actionable $action$.</p> <p>-The next section will show that induction by analogy is actually occurring in this process.</p> <p>- Applies to the CCA5 (Figure 1), CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture; in the CCA5 version of the architecture its sole Navigation Module should be considered the same as Navigation Module A</p>

Table A19. Summary of the Analogical Feedback Operations from the Navigation Module A per Equations (88a) – (94)

A.10 Induction by Analogy

This section of Appendix A applies to the CCA5 (Figure 1), CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture. In the CCA5 version of the architecture its sole Navigation Module should be considered the same as Navigation Module A.

Below it is shown that the movement and comparison of navigation maps in equations (90 – 94) allows induction by analogy to occur. These equations are re-written using the more descriptive terms for the Working Navigation Maps used in Table A19—to distinguish between the several different navigation maps which are considered the current Working Navigation Map **WNM'** at some point, descriptive suffixes are added to **WNM'**. These descriptive suffixes are for the reader, they do not exist in the architecture.

$$\begin{aligned}
 &((action_t \neq \text{"move*"} \text{ or } \mathbf{WPR}_t = [\text{"analogical*"}]) \text{ and } \mathbf{WPR}_t \neq [\text{"discard*"}] \text{ and } \mathbf{WPR}_t \neq [\text{"feedback*"}]), \\
 &\Rightarrow \text{Nav_ModA.feedback_to_assocn_mod}(\mathbf{WNM}'_{t-original}) \quad (95) \text{ (from 90)} \\
 &\Rightarrow \mathbf{WNM}'_{t-best_match} = \text{Causal_Mem_Mod.match_best_map}(\mathbf{WNM}'_{t-original}) \quad (96) \text{ (from 91)} \\
 &\Rightarrow \mathbf{TempMap}_t = \text{Nav_ModA.use_linkaddress1_map}(\mathbf{WNM}'_{t-best_match}) \quad (97) \text{ (from 92)} \\
 &\Rightarrow \mathbf{WNM}'_{t-difference} = \text{Nav_ModA.subtract}(\mathbf{WNM}'_{t-best_match}, \mathbf{TempMap}_t) \quad (98) \text{ (from 93)}
 \end{aligned}$$

$$\begin{aligned}
 &((action_{t-1} \neq \text{"move*"} \text{ or } \mathbf{WPR}_{t-1} = [\text{"analogical*"}]) \text{ and } \mathbf{WPR}_{t-1} \neq [\text{"discard*"}] \text{ and } \mathbf{WPR}_{t-1} \neq [\text{"feedback*"}]), \\
 &\Rightarrow \mathbf{WNM}'_{t-analogical} = \text{Nav_ModA.retrieve_and_add_vector_assocn}() \quad (99) \text{ (from 94)}
 \end{aligned}$$

Consider a definition of induction by analogy. There are two variables **x** and **y**. Variable **x** has properties $P_1, P_2, P_3, P_4, \dots P_n$ (100). Variable **y** also has properties $P_1, P_2, P_3, P_4, \dots P_n$ (101). It happens that variable **y** has another property **N** (102). Therefore in (103) it can be concluded by induction by analogy that variable **x** also has property **N**.

In (95) consider $\mathbf{WNM}'_{t-original}$ as variable **x**, or perhaps as navigation map **x**. It is desired to know what this navigation map **x** will do next, i.e., which navigation map will it call. Consider variable **y**, or perhaps named as navigation map **y**, as referring to (96) $\mathbf{WNM}'_{t-best_match}$. It is the best matching navigation map to navigation map **x** and thus assumed it will share many properties. Next there is exploration of what navigation map **y** does next (i.e., what navigation map does the *linkaddress* chosen link to). Navigation map **y** calls the navigation map in **TempMap** (97) and that the difference between navigation map **y** and **TempMap** is $\mathbf{WNM}'_{t-difference}$ (98). Thus, consider this difference, i.e., $\mathbf{WNM}'_{t-difference}$ to be property **N**. Since navigation map **y** has property **N**, therefore by induction by analogy, it can be said that navigation map **x** also has property **N** (103). Thus, add property **N**, which is actually $\mathbf{WNM}'_{t-difference}$, to navigation map **x**, which is actually $\mathbf{WNM}'_{t-original}$, producing the result of navigation map **x** with property **N** as being $\mathbf{WNM}'_{t-analogical}$ (99).

$$P_1x \ \& \ P_2x \ \& \ \dots \ P_nx \quad (100)$$

$$P_1y \ \& \ P_2y \ \& \ \dots \ P_ny \quad (101)$$

$$N_y \quad (102)$$

$$\therefore Nx \quad \square \quad (103)$$

It is important to note that analogical problem solving is not a separate module in the architecture (for example, to be used when solving intelligence tests or difficult problems) but rather part of the core mechanism of day-to-day functioning of the architecture. These equations remain largely unchanged in the CCA6 architecture from the prior CCA5 architecture although there are small modifications to account for the duplicate Navigation Module. Note that equations (i) to (ix) in the text of paper above are easy-to-read versions of equations (95) to (103).

$((action_t \neq \text{"move*"} \text{ or } WPR_t = [\text{"analogical*"}]) \text{ and } WPR_t \neq [\text{"discard*"}] \text{ and } WPR_t \neq [\text{"feedback*"}]),$ \Rightarrow	-see Table A18 -see Figure 5 (CCA6 version) or Figure 9 (CCA7 version of the architecture) for overview of modules and interconnections
$\Rightarrow Nav_ModA.feedback_to_assocn_mod(WNM'_{t-original})$ (95)	-see Table A18 - $WNM'_{t-original}$ – the current Working Navigation Map WNM'_t which is given the label 'original' for better understanding of what is happening
$\Rightarrow Causal_Mem_Mod.match_best_map(WNM'_{t-original})$ (96) $\rightarrow WNM'_{t-best_match}$	-see Table A18 $WNM'_{t-original}$ – the current Working Navigation Map WNM'_t which is given the label 'original' for better understanding of what is happening
$\Rightarrow Nav_ModA.use_linkaddress1_map(WNM'_{t-best_match})$ (97) $\rightarrow TempMap_t$	-see Table A18 - WNM'_{t-best_match} – the current Working Navigation Map WNM'_t which is given the label 'best-match' for better understanding of what is happening
$\Rightarrow Nav_ModA.subtract(WNM'_{t-best_match}, TempMap_t)$ (98) $\rightarrow WNM'_{t-difference}$	-see Table A18 - $WNM'_{t-difference}$ – the current Working Navigation Map WNM'_t which is given the label 'difference' for better understanding of what is happening
$((action_{t-1} \neq \text{"move*"} \text{ or } WPR_{t-1} = [\text{"analogical*"}]) \text{ and } WPR_{t-1} \neq [\text{"discard*"}] \text{ and } WPR_{t-1} \neq [\text{"feedback*"}]), \Rightarrow$	-see Table A18
$\Rightarrow WNM'_{t-analogical} = Nav_ModA.retrieve_and_add_vector_assocn()$ (99) $\rightarrow WNM'_{t-analogical}$	-see Table A18 - $WNM'_{t-analogical}$ – the current Working Navigation Map WNM'_t which is given the label 'analogical' for better understanding of what is happening
$P_{1x} \& P_{2x} \& \dots P_{nx}$ (100)	-explanation of definition of induction by analogy - variable (or navigation map) x has properties $P_1, P_2, P_3, P_4, \dots P_n$
$P_{1y} \& P_{2y} \& \dots P_{ny}$ (101)	-explanation of definition of induction by analogy -variable (or navigation map) y also has properties $P_1, P_2, P_3, P_4, \dots P_n$
N_y (102)	-explanation of definition of induction by analogy

	- variable (or navigation map) y also has property N
$\therefore N_x \square (103)$	-explanation of definition of induction by analogy - thus, can conclude by induction by analogy that variable (navigation map) x also has property N

Table A20. Explanation of Symbols and Pseudocode in Equations (95) – (103)

Input:	<i>not applicable as this is a continuation of Navigation Module A operations</i>
Output:	- WNM' _{r-analogical} (99)
Description:	<p>-See Table A18. Equations (95 – 99) are re-write of equations (90 – 94) using the more descriptive terms for the Working Navigation Maps. These descriptive suffixes are for the reader, they do not exist in the architecture.</p> <p>-To distinguish between the several different navigation maps which are considered the current Working Navigation Map WNM' at some point, descriptive suffixes are added to WNM' listed above.</p> <p>- Equations (100 – 103) consider a definition of induction by analogy applied to the operations performed on the Working Navigation Maps.</p> <p>- Applies to the CCA5 (Figure 1), CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture; in the CCA5 version of the architecture its sole Navigation Module should be considered the same as Navigation Module A</p>

Table A21. Summary of the Analogical Feedback Operations from the Navigation Module A per Equations (95) – (103)

A.11 Grounding

This section of Appendix A applies to the CCA5 (shown in Figure 1), the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

Like the prior CCA5 architecture, the CCA6 architecture provides a solution to the grounding problem—information enters the system through experiential sensory systems. Information will automatically be mapped to real-world sensations and actions. No information is kept as an isolated symbol—*everything* in the CCA6 is within a navigation map linked to other navigation maps.

A *feature* was defined initially in (34). Cells in navigation maps can contain *features*, *procedures*, or *linkaddresses*. Below equation (104) defines a *grounded_feature* as being any *feature* such that the *feature* is in some sensory system Local Navigation Map *all_LNMs_x* and that the *feature* also was present in some sensory system array **S**_{σ,t} (1–7) represented by *s(t)* (9). This implies that the CCA6 experientially acquired this *feature*, and it is in its navigation maps associated with other sensory

inputs and possibly associated with previous or upcoming *features*. The symbol grounding problem is largely resolved by grounding symbols with their real-world sensations, interactions, and linkages.

However, the second part of equation (104) deals with the case where information has been acquired by electronic transfer or by the equivalent of human memorization, or possibly an experiential acquisition of the *feature* was poor, and grounding is poor. A *feature* is considered as a *grounded_feature* if the *feature* is present in a Working Navigation Map \mathbf{WNM}'_t and in the previous cognitive cycle the analogical feedback mechanism was used. Note that if in the previous cognitive cycle ($t-1$) the action was not actionable, i.e., $action_{t-1} \neq \text{"move*"}$, then the analogical feedback mechanism will occur, or if the previous Working Primitive \mathbf{WPR}_{t-1} specified an analogical feedback loop, then the analogical feedback mechanism will occur.

Equation (105) essentially states that any cell in any of the addressable navigation maps (e.g., it will not include, for example, the specialized navigation maps in the Sequential/Error Correcting Module) will either contain a *grounded_feature* or contain a *linkaddress* pointing to another cell which most likely will be in another navigation map, or else the cell has no features.

With regard to accepting as grounded a *linkaddress* pointing to another cell/navigation map, it is assumed that there are so many links pointing to so many other navigation maps in a chain of *linkaddresses* that if a cell points to another cell most likely in another navigation map, it will be grounded there by association with some valid grounding condition (e.g., sensations, actions, etc.) or else point to another navigation map where it will be ground there, and so on. There may be differences in the quality of the grounding as such, but it will always be grounded to some extent.

These equations remain largely unchanged in the CCA6 architecture from the prior CCA5 architecture.

$$\begin{aligned} \text{grounded_feature} = & \forall_{\text{feature}} : (\text{feature} \in \text{all_LNMs}_x \text{ AND } \text{feature} \in s(t)) \\ & \text{OR} \\ & \forall_{\text{feature}} : ((\text{feature} \in \mathbf{WNM}'_t \text{ AND } \text{action}_{t-1} \neq \text{"move*" AND } \mathbf{WPR}_{t-1} \neq [\text{"feedback*"}]) \text{ OR} \\ & \mathbf{WPR}_{t-1} = [\text{"analogical*"}]) \end{aligned} \quad (104)$$

$$\forall_{x,t} : \text{all_navmaps}_{x,t} = \text{grounded_feature} \text{ OR } \text{link}(\text{all_navmaps}_{x,t}) \neq [] \text{ OR } \text{cellfeatures}_{x,t} = [] \quad (105)$$

<i>feature</i>	<ul style="list-style-type: none"> - $feature \in \mathbb{R}$ (34) - some arbitrary real number representing a <i>feature</i> modality and value - for example, $feature_{13,(7modcode=*Causal_Memory_Mod*, mapno=3456,2,3,4)}$ would be <i>feature</i> number 13 in the cell $x=2,y=3,z=4$ in map number 3456 in the Causal Memory Module; its value, for example, could represent a visual line
----------------	---

<i>grounded_feature</i>	<p>- any <i>feature</i> such that the <i>feature</i> is in some sensory system Local Navigation Map <i>all_LNMs_χ</i> and that the <i>feature</i> also was present in some sensory system array <i>S_{σ,t}</i> (1–7) represented by <i>s(t)</i> (9)</p> <p>- or if the <i>feature</i> has been acquired by electronic transfer or by the equivalent of human memorization then consider a <i>feature</i> as a <i>grounded_feature</i> if the <i>feature</i> is present in a Working Navigation Map <i>WNM'_t</i> and in the previous cognitive cycle the analogical feedback mechanism was used</p> <p>-see Figure 1(CCA5 version) Figure 5 (CCA6 version) or Figure 9 (CCA7 version of the architecture) for overview of modules and interconnections</p>
$\forall_{feature}: (feature \in all_LNMs_{\chi} \text{ AND } feature \in s(t))$ $\rightarrow grounded_feature$	<p>- any <i>feature</i> such that the <i>feature</i> is in some sensory system Local Navigation Map <i>all_LNMs_χ</i> and that the <i>feature</i> also was present in some sensory system array <i>S_{σ,t}</i> (1–7) represented by <i>s(t)</i> (9)</p>
$\forall_{feature}: ((feature \in WNM'_t \text{ AND } action_{t-1} \neq \text{"move*"} \text{ AND } WPR_{t-1} \neq [\text{"feedback*"}]) \text{ OR } WPR_{t-1} = [\text{"analogical*"}])$ $\rightarrow grounded_feature$	<p>- for example, a <i>feature</i> that has been acquired by electronic transfer or by the equivalent of human memorization, then the analogical feedback mechanism will occur (when the <i>feature</i> is considered), and consider that <i>feature</i> as a <i>grounded_feature</i></p> <p>- although the older feedback mechanism <i>may</i> result in grounding the feature adequately, often it won't, thus requiring utilization of the analogical feedback mechanism to ensure grounding</p>
<i>cellfeatures_{χ,t}</i> = []	<p>-<i>cellfeatures_{χ,t}</i> are all the features within a given cell <i>χ</i> (38)</p> <p>-thus, <i>cellfeatures_{χ,t}</i> = [] means there are no features in that cell at address <i>χ</i></p>
$\forall_{\chi,t}: all_navmaps_{\chi,t} = grounded_feature$ OR $link(all_navmaps_{\chi,t}) \neq []$ OR <i>cellfeatures_{χ,t}</i> = []	<p>- any cell in any of the addressable navigation maps will either contain a <i>grounded_feature</i>, or contain a <i>linkaddress</i> pointing to another cell which most likely will be in another navigation map, or else the cell has no features</p>

Table A22. Explanation of Symbols and Pseudocode in Equations (104) – (105)

Input:	<i>not applicable as this is a continuation of Navigation Module A operations</i>
Output:	<i>not applicable as this is a continuation of Navigation Module A operations</i>
Description:	-These equations specify that for every addressable navigation map in the Causal Cognitive Architecture <i>features</i> can be considered to be grounded or to be used in a grounded fashion.

	- Applies to the CCA5 (shown in Figure 1), the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the architecture
--	---

Table A23. Summary of the Grounding Requirements with the Architecture per Equations (104) – (105)

A.12 Simple Language Generation

This section of Appendix A applies to the CCA6 (Figure 5) and the CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

The generation of simple language (i.e., as opposed to full compositional language) was discussed in the previous Causal Cognitive Architectures, including the previous CCA5 one. To a large extent, some way of communicating what is in a navigation map(s) from one CCA6 (or CCA7) embodiment to another CCA6(or CCA7) embodiment is required. In the CCA5 architecture this was accomplished with a straightforward instinctive primitive named `apply_primitive_nav_to_protolang()`.

In the CCA6 or CCA7 version of the architecture, while compositional language comprehension and behavior makes use of Navigation Module B, as was illustrated in the text and figures of the main body of the paper, language expression remains simple (i.e., essentially a simple read-back of what is on the navigation maps is being communicated) and still remains generated by Navigation Module A. In future versions of the architecture, full compositional generation of language is expected.

In (106) `Nav_ModA.apply_primitive_nav_to_protolang(WPRt, WNM't)` produces an output action *action*_{*t*}. As before, per equations (83 – 87), *action*_{*t*} is propagated from the Navigation Module A to the Output Vector Association Module A. As before, there is motion correction of the output signal with interactions with the Sequential/Error Correcting Module, and then the output signal is sent to the Output Vector Association Module A and then on to the actual output actuators.

The equation below remains largely unchanged in the CCA6 or CCA7 versions of architecture from the prior CCA5 architecture, although it now specifies the Navigation Module being used.

$$action_t = \text{Nav_ModA.apply_primitive_nav_to_protolang}(\mathbf{WPR}_t, \mathbf{WNM}'_t) \quad (106)$$

<code>Nav_ModA.apply_primitive_nav_to_protolang(WPR_{<i>t</i>}, WNM'_{<i>t</i>})</code> $\rightarrow action_t$	-Nav_Mod refers to the Navigation Module A (Figure 5 or Figure 9) - <code>apply_primitive_nav_to_protolang(WPR_{<i>t</i>}, WNM'_{<i>t</i>})</code> is pseudocode for an algorithm which applies the Working Primitive WPR _{<i>t</i>} to the Working Navigation Map WNM' , such that the current WNM' and sequence of WNM' Working Navigation Maps preceding it (i.e., what happened? why did it
---	---

	<p>happen?) are converted into a simple proto-language which is represented in the $action_t$ value</p> <ul style="list-style-type: none"> -this is a simple read-back (limited verbs used) of the navigation maps - the $action_t$ value will be transformed into an output signal that activates the appropriate actuators to communicate this simple proto-language communication to another CCA6 embodiment or a human - see Table A14 for more details of converting WPR_t and WNM'_t into an actionable output, which in this case represents a communication
--	--

Table A24. Explanation of Symbols and Pseudocode in Equation (106)

Input:	<p>WPR_t : the Working Primitive WPR is the primitive that will be applied against the Working Navigation Map WNM' in the Navigation Module A, and with regard to (106) will involve transforming the recently stored (in the Causal Memory Module) Working Navigation Maps WNM's into an explanation and communication of what happened and why it happened, as well as possibly communication of actions, requests, or concepts</p> <p>WNM'_t : the current Working Navigation Map which is derived from the processed sensory inputs or analogic feedback results</p>
Output:	<p>$action_t$: a signal to move some actuator or send an electronic signal e.g., <move right> e.g., <sound 2000Hz> it is sent to the Output Vector Association Module A where more detailed instructions are created to effect the required actuator outputs</p>
Description:	<p>-The Navigation Module A applies the Working Primitive WPR_t against the current Working Navigation Map WNM' in the Navigation Module A, and with regard to (106) will involve transforming the recently stored (in the Causal Memory Module) Working Navigation Maps WNM's into an explanation and communication of what happened and why it happened, as well as possibly communication of actions, requests, or concepts.</p> <p>-Applies to the CCA6 or CCA7 versions of the architecture</p>

Table A25. Summary of the Operation of Simple (Non-Compositional) Language Generation per Equation (106)

A.13 Navigation Module B

This section of Appendix A applies to the CCA6 (Figure 5) and CCA7 (Figure 9) versions of the Causal Cognitive Architecture.

As seen in Figure 5 there are now in the CCA6 two Navigation Modules—Navigation Module A and Navigation Module B (as opposed to the single Navigation Module in the prior CCA5 architecture, as seen in Figure 1). (This largely holds for the CCA7 version of the architecture although in this version there are multiple Navigation Module B's. However, in this section, one of the Navigation Module B's can be considered.)

Similar to Navigation Module A, in Navigation Module B an instinctive primitive or a learned primitive (which themselves are navigation maps) in conjunction with the Working Navigation Map **WNMB'** (similar to the Working Navigation Map **WNM'** that Navigation Module A operates on) may result in an action signal. This signal goes to the Output Vector Association Module B and to the external embodiment (Figure 5).

Working Navigation Map B **WNMB'** (107) is defined similarly to Working Navigation Map A **WNM'**—both are navigation maps, i.e., treated as arrays. Similarly, as equation (108) shows, any operation that in theory Navigation Module A can perform on a Working Navigation Map, Navigation Module B can also perform a similar operation on a similar Working Navigation Map. This implies that equations (1) to (106) above all also apply to the operations Navigation Module B on Working Navigation Map B **WNMB'**. Indeed, Navigation Module B originates from the duplication of Navigation Module A. However, the duplication is assumed to be somewhat of a partial duplication (not all pathways would have duplicated similarly necessarily) and in the architecture described in the main body of the text above, Navigation Module B is used mainly for compositional processing. Hence, in (108) there are what are called *navA_only_operation's*—operations which are only performed by Navigation Module A.

While in the architecture described in the main body of the text above, Navigation Module B is used mainly for compositional language comprehension and resultant compositional behavior, in fact, Navigation Module B can be used for much more abstract compositional comparisons of two sets of data in the two Navigation Modules. This is beyond the scope of the present paper. The set of *navA_only_operation's* can readily be adjusted in different versions of the architecture.

Consider a demonstration example shown above in Figure 6A. There are a number of shape objects in the sensory scene plus the instruction “place the black sphere on top of the black block which is not near a cylinder.” As described by equations (1) to (106) above, the objects in the sensory scene will become mapped/matched/updated onto the **WNM'** navigation map in Navigation Module A. The instruction, however, will trigger the instinctive primitive `parse_sentence()` (which is broken down in sub-parts in equations (109) to (111)).

Equation (109) shows that if a sensory input is recognized as an instruction sentence *instruction_sentence* then the instinctive primitive `Nav_ModB.parse_sentence.copy()` maps the instruction sentence onto Working Navigation Map B **WNMB'** in Navigation Module B.

The instinctive primitive `Nav_ModB.parse_sentence.parse()` then parses through the instruction sentence, i.e., Navigation Map B **WNMB'** in Navigation Module B (110). For each word in the instruction sentence `Nav_ModB.parse_sentence.parse.match()` matches the word against the Causal Memory Module (111). If an action word is found then it is mapped to cells in Navigation Map A **WNM'** in Navigation Module A which contain matching features to the cells in the instruction sentence (i.e., Navigation Map B **WNMB'** in Navigation Module B) associated with the action word.

Equation (112) shows that if a *near_trigger* occurs in parsing, then the instinctive primitive `Nav_ModB.physics_near_object()` is triggered. A *near_trigger* is a word (or other sensory input that does the same thing) related to the geometric concepts of near and not near. Continuing the compositional example, the word “near” acts as a *near_trigger* (Navigation Module B). Since there was a “not” beside it, and then “cylinder” this instinctive primitive flags the cell in Navigation Module A (i.e., Navigation Map A **WNM'**) with “cylinder” with the flag “not”. Also, the

`physics_near_object()` instinctive primitive will mark “not” in the matched cell and all the adjoining non-null cells.

Equation (113) shows that once the instinctive primitive `Nav_ModB.parse_sentence()` has reached the end of the instruction or communication sentence, i.e., there is nothing left to parse, then it looks for cells which have been tagged or marked with tags. Possible associated tags depend on the collection of instinctive primitives, which at present only has the single option of `place_object()` (but thousands of instinctive and learned primitives are possible here).

Given a `<place>` tag in cell (0,0,0) of Navigation Module A in Figure 7B in the example in the text, the instinctive primitive `Nav_ModA.place_object()` will be triggered (113). Note that this instinctive primitive will operate on Navigation Module A (i.e., Navigation Map A **WNM'** which is the representation of the sensory scene.

The `Nav_ModA.place_object()` instinctive primitive will attempt to move and place an object somewhere. Continuing the example above, the tag `<“place”>` is most closely associated with “sphere, black”—that is the object that will be placed somewhere. The primitive now looks for other tagged notations such as `<“top”>` in the navigation map (Figure 7B). It will see a `<“not”>` in the cell with the features of a black block (Schneider, 2024) at (2,0,0) and not consider the `<“top”>` tag in that cell. However, the cell with the features of black block at (4,0,0) has a valid tag such as `<“top”>`. Thus, this primitive will in turn trigger the instinctive primitive `Nav_ModA.move()` to perform the movement of the black sphere to cell (4,0,0) (on top of it if three dimensions were being used) (114).

`Nav_ModA.move()` can be used for many actions of the CCA6. In this example, actions from Navigation Module A cause an action signal to go to the Output Vector Association Module A (Figure 5). The Output Vector Association Module A performs motion planning and motion corrections via feedback from the Sequential/Error Correcting Module. The corrected motion action signal from the Output Vector Association Module A is shown in Figure 5. This signal goes to the Output Vector Shaping Module which produces the actual signals (Figure 5) for actuators to move the black sphere to the cell with the black block on the right. Thus, the black sphere is then moved to (and on top of it in 3 dimensions) the black block on the right side.

Then a new cognitive cycle repeats again, finally processing again the actual sensory inputs streaming into the architecture.

$$\mathbf{WNMB}' = \in \mathbb{R}^{m \times n \times o \times p} \quad (107)$$

$$\begin{aligned} & \forall operation_{NavModA}(), \\ & (\sim navA_only_operation \Rightarrow operation_{NavModB}(\mathbf{WNMB}_m) = operation_{NavModA}(\mathbf{WNMB}_m)) \quad (108) \\ & (instruction_sentence), \\ & \Rightarrow \mathbf{WNMB}'_i = Nav_ModB.parse_sentence.copy() \quad (109) \\ & \Rightarrow Nav_ModB.parse_sentence.parse(\mathbf{WNMB}'_i), \quad (110) \\ & \quad \Rightarrow Nav_ModB.parse_sentence.parse.match() \quad (111) \\ & \quad \Rightarrow near_trigger, \\ & \quad \quad \Rightarrow Nav_ModB.physics_near_object() \quad (112) \\ & \Rightarrow end_of_communication, \\ & \quad <place>, \end{aligned}$$

$\Rightarrow \text{Nav_ModA.place_object}() \quad (113)$
 $\Rightarrow \text{Nav_ModA.move}() \quad (114)$

WNMB'	<p>-Working Navigation Map B</p> <p>-the current Working Navigation Map B WNMB'_t is the navigation map which the Navigation Module B focuses its attention on, i.e., applies operations on to make a decision to take some sort or no action</p>
$\forall \text{operation}_{\text{NavModA}}(\mathbf{WNM}_m),$ $(\sim \text{navA_only_operation} \Rightarrow \text{operation}_{\text{NavModB}}(\mathbf{WNM}_m) = \text{operation}_{\text{NavModA}}(\mathbf{WNM}_m))$	<p>-any operation that in theory Navigation Module A can perform on some Working Navigation Map m in Navigation Module A \mathbf{WNM}_m ($\text{operation}_{\text{NavModA}}(\mathbf{WNM}_m)$), Navigation Module B can also perform a similar operation on a similar Working Navigation Map m \mathbf{WNM}_m in Navigation Module B</p> <p>-implies that equations (1) to (106) above all also apply to the operations of Navigation Module B on Working Navigation Map B WNMB'</p> <p>-<i>navA_only_operation</i>'s—operations which are only performed by Navigation Module A; set of these nav_modA-only operations can be adjusted for different versions of the architecture</p> <p>-in CCA6 and CCA7 versions of the architecture, Navigation Module B largely used for compositional language comprehension and behavior</p>
<i>instruction_sentence</i>	-a sensory input which triggers recognition as an input instruction sentence by the Input Sensory Vectors Association Module and/or Instinctive Primitives and/or Learned Primitives
WNMB' _t = Nav_ModB. parse_sentence.copy()	-instinctive primitive <code>parse_sentence.copy()</code> maps the <i>instruction_sentence</i> instruction sentence onto Working Navigation Map B WNMB' in Navigation Module B
Nav_ModB.parse_sentence. parse(WNMB' _t)	-instinctive primitive <code>parse_sentence.parses()</code> parses through the instruction sentence, i.e., Navigation Map B WNMB' in Navigation Module B
Nav_ModB.parse_sentence. parse.match()	<p>-for each word in the instruction sentence Nav_ModB. <code>parse_sentence.parse.match()</code> matches the word against the Causal Memory Module</p> <p>- if an action word is found then it is mapped to the Navigation Map A WNM' in Navigation Module A.</p>
<i>near_trigger</i>	<p>-word related to positional relationships of near and not near</p> <p>-triggers instinctive primitive <code>physics_near_object()</code></p>
<i>near_trigger</i> , $\Rightarrow \text{Nav_ModB.}$ <code>physics_near_object()</code>	-if a <i>near_trigger</i> occurs in parsing, then the instinctive primitive <code>Nav_ModB.physics_near_object()</code> is triggered

	-the <code>Nav_ModB.physics_near_object()</code> will mark “near” (or “not” if “not” is specified in the instruction) the cell, in Navigation Map A WNM' in Navigation Module A, which matches to this instinctive primitive, as well as all the adjoining non-null cells.
<code>end_of_communication</code>	-end of a language sentence, i.e., nothing left to parse
<code><place></code>	-existence of a <code><place></code> tag
<code>end_of_communication,</code> <code><place>,</code> <code>⇒ Nav_ModA.</code> <code>place_object()</code>	<p>-once the instinctive primitive <code>Nav_ModB.parse_sentence()</code> has reached the end of the instruction or communication sentence, i.e., there is nothing left to parse, then it will trigger associated tags</p> <p>-possible associated tags depend on the collection of instinctive primitives, which at present only has the single option of <code>place_object()</code> (but thousands of instinctive and learned primitives are possible here)</p> <p>-given a <code><place></code> tag in cell (0,0,0) of Navigation Module A in Figure 5 in the example in the text, the instinctive primitive <code>Nav_ModA.place_object()</code> is triggered</p> <p>- Note that this instinctive primitive will operate on Navigation Module A (i.e., Navigation Map A WNM')</p> <p>- The <code>Nav_ModA.place_object()</code> instinctive primitive will attempt to move and place an object somewhere depending on what tags the other instinctive primitives <code>Nav_ModB.parse_sentence</code>.</p> <p><code>parse.match()</code> and <code>Nav_ModB.physics_near_object()</code> have written to Navigation Map A WNM'</p>
<code>Nav_ModA.move()</code>	-once <code>Nav_ModA.place_object()</code> has a location to move the object to, <code>Nav_ModA.move()</code> is triggered and actually sends an action signal to the Output Vector Association Module A which then outputs a more processed action signal to other modules resulting in the architecture's output actuators being instructed to move the object

Table A26. Explanation of Symbols and Pseudocode in Equations (107) – (114)

Input:	<p>WNM'_t: the current Working Navigation Map A in Navigation Module A which is derived from the processed sensory inputs representing the sensory scene</p> <p>WNMB'_t: the current Working Navigation Map B in Navigation Module B which is derived from a communication sentence</p>
---------------	---

	<p>WPR: various Working Primitives WPR's are the primitives that will be applied against the Working Navigation Map A WNM' in the Navigation Module A and/or Working Navigation Map B WNMB' in the Navigation Module B; these include:</p> <ul style="list-style-type: none"> - Nav_ModB.parse_sentence() - Nav_ModB.physics_near_object() - Nav_ModA.place_object() - Nav_ModA.move()
Output:	<p><i>action</i>: a signal to move some actuator or send an electronic signal</p> <p>-it is sent from Navigation Module A to the Output Vector Association Module A (Figure 5) where more detailed instructions are created to effect the required actuator outputs to the Output Vector Shaping Module (1) to the actual real-world actuators</p>
Description:	<p>-A communication or instruction sentence is parsed in Navigation Module B and applied against a relevant represented sensory scene in Navigation Module A. As such, the beginnings of compositional language comprehension and behavior emerge.</p> <p>- Applies to the CCA6 (Figure 5) and CCA7 (Figure 9) versions of the Causal Cognitive Architecture; however, there are multiple Navigation Module B's in the CCA7 version, thus one of the Navigation Module B's can be considered</p>

Table A27. Summary of the Operation of Compositional Language Comprehension per Equations (107) – (114)

A.14 Enhanced Navigation Module B's

This section only applies to the CCA7 version of the architecture.

Consider an agent, i.e., a robot, controlled by the CCA7 architecture shown in Figure 9. For simplicity the CCA7 architecture and the robot embodiment will be called the “CCA7” or “CCA7 robot.” The CCA7 robot comes to location “X” in Figure 10. It receives the instruction that starting at its existing position (i.e., “X”) it must visit each object and then return to the starting location.

While in location “X” the CCA7 robot maps a sensory scene into the navigation map in Navigation Module A, which is what it automatically does in each available cognitive cycle when there are new sensory inputs to process. The resulting navigation map in Navigation Module A is shown in Figure 11A. The CCA7 robot receives distances (either with the visual sensory information or via a separate ultrasonic distance sensory system). The numbers refer to the distance (in centimeters) between the objects in the different cells. (The distance number can be determined by matching the same number in the path between two cells. As well, note a clockwork recording of distances in each cell.) The instruction “go to all objects and go back” is placed in Navigation Module B as shown in Figure 11B. These operations are similar in nature to ones already described above for the CCA6 version of the architecture in its initial processing of the example of the sensory scene and instruction concerning the “placing a black sphere on top of the black block which is not near a cylinder” (Figure 6, Figure 7).

However, as described above and shown in Figure 9, there are now in the CCA7 multiple Navigation Modules—one Navigation Module A and over a thousand (1024) duplicated Navigation Module B's.

Equation (115) (taken from Appendix A in the Supplementary Material section) indicates that the Working Navigation Map B' **WNMB**' (upon which primitives operate in Navigation Module B) is an array like before, but now can be one of 1024 different navigation maps (corresponding to a different navigation map in each of the Navigation Module B's.)

The Navigation Module B's are numbered $n=1$ to $n=1023$. The top (or first) Navigation Module B appears to be the $n=1$ Navigation Module B, as shown in Figure 9. However, a $n=0$ Navigation Module B exists and is used to store a copy of the compositional instructions so that if the other layers are overwritten, there is still a copy of the instructions. Layer $n=0$ is considered "reserved" and will not be overwritten. If there is other information that an instinctive or learned primitive needs to ensure remains intact for the current operations, other Navigation Module B's can be temporarily designated "reserved" as well.

Equation (116) indicates that the same instinctive primitive or the same learned primitive is initially applied to all of the Navigation Module B's. (In subsequent cognitive cycles the initial primitive applied may trigger different primitives in different Navigation Modules.) If an instinctive primitive or learned primitive does not examine processing paths in parallel, then it will simply use the results of operations in Navigation Module B $n=1$ and ignore the other Navigation Module B's.

As shown in Figure 11A is Navigation Module A containing Working Navigation Map (**WNMA**) of the sensory scene of the various places the agent has to navigate to. In Navigation Module B $n=1$ (Figure 11B) is a Working Navigation Map (**WNMB_{n=1}**) of the instruction sentence to "go to all the objects and go back."

The word "go" in the first cell of the navigation map in Navigation Module B (Figure 11A) is matched against the Causal Memory Module as an action word and triggers the instinctive primitive "goto()" (117). "**WNMA**'_t = Nav_ModA.goto()" indicates that this instinctive primitive, i.e., "goto()" is being applied to the Working Navigation Map A in Navigation Module A.

The instinctive primitive "goto()" causes the CCA7 robot to tag a location(s) and then essentially move to whatever location is indicated by the tag(s). The word "all" which is associated with active word <"go"> (until another action word is encountered, as in an earlier example above) will cause the tag <"all"> to be placed in all the cells with objects in the navigation map in Navigation Module A (Figure 12A).

The words "go back" is also associated with the instruction word <"go"> and will cause the tag <"back"> to be placed in the starting cell (which is (0,0,0) in this example). This can be seen in Figure 12A.

Once the instinctive primitive "goto()" tags the cell(s) where it has to move to, it then decides if it will move (i.e., "go to") the cell with the tag. However, if there are multiple tags, i.e., multiple locations to navigate to ("locations > 1" in (118–123)), then the "small_plan()" instinctive primitive is activated instead of moving to a single location. As discussed above, this instinctive primitive will plan a navigation route to whatever multiple tagged locations are indicated on the navigation map(s).

Once activated (118), the instinctive primitive "small_plan()" (regardless of argument) will copy the Navigation Map A to all "non-reserved" Navigation Module B's, i.e., $n=1 \dots 1023$ in this example. It will remove any action words such as "go" in the example above. This copying is

indicated by the arrow in Figure 12. Thus, the instruction sentence in Navigation Module B $n=1$ (Figure 11) is overwritten here. (Navigation Module B $n=0$ is “reserved” for a copy of the instruction sentence, although it actually will not be used again in this example.) A number of existing operations in various instinctive primitives already transfer or compare the contents of Navigation Module A and Navigation Module B with each other. Thus, the emergence of this step is a feasible one in the continued evolution of the architecture.

Equation (118) describes instinctive primitive “`small_plan(random=False)`” acting on the Working Navigation Map (**WNMB**, $n=1$) in Navigation Module B $n=1$ ($\text{Nav_ModB}_{n=1}$) at time $t=t$ (i.e., at some specific time). The instinctive primitive “`small_plan(random=False)`” follows nearest neighbour algorithm discussed above. In making a plan where to navigate to, this primitive will choose the tag (i.e., location) that is closest to tag (i.e., location) from where it is navigating. The argument `random=False` indicates that this instinctive primitive does not introduce any random variations. As will be seen below, in the other Navigation Module B’s $n=2\dots1023$ random variations will be introduced.

In $n=1$ Navigation Module B, “`small_plan(random=False)`” operates on the navigation map shown in Figure 12b and decides which tagged cell to navigate first. It uses a nearest neighbor algorithm in its planning actions. For example, in Figure 12B, the cell (0,0,0) in which the CCA7 is starting from, has a distance of 25, 31, and 42 units (actually centimeters, but “`small_plan()`” will disregard the actual units) to the other objects. (They are listed as “25, 42, 31” in cell (0,0,0) in Figure 12B due to a clockwise organization of distances.) According to the nearest neighbor algorithm it chooses the shortest distance, which is 25, i.e., it plans to navigate first to cell (2,2,0) containing the white sphere. Thus, it changes the <“all”> tag to a <1>. This can be seen in Figure 12C.

The instinctive primitive then considers navigating from cell (2,2,0)—which object to navigate to next? As can be seen in Figure 12B or Figure 12C, 22 is the shortest distance, thus it decides to navigate to cell (3,0,0) which contains the black sphere. It changes the <“all”> tag to a <2>. It then considers navigating from cell (3,0,0)—which object to navigate to next? Actually, the only untagged object remaining is the white block in cell (4,2,0), which is then tagged with a <3>. This is shown in Figure 12C.

If this was the previous CCA6 version of the architecture (albeit, retrofitted with these new equations) with only one Navigation Module B, then at this point the instinctive primitive “`small_plan(random=False)`” would trigger the instinctive primitive “`move()`” to move a CCA6 robot to cell (2,2,0) containing the white sphere. Then the instinctive primitive “`move()`” is triggered again to move to cell (3,0,0) containing the black sphere. Then the instinctive primitive “`move()`” is triggered again to move to cell (4,2,0) containing the white block. Then the instinctive primitive “`move()`” is triggered again to move to cell (0,0,0) which was the starting point. From Figure 12C, note that the sum of the distances is $25+22+20+42=109$ cm in this navigation route.

However, in the CCA7 version of the architecture being considered here, there are over a thousand Navigation Module B’s. As equation (120) indicates, for Navigation Module B $n=2\dots1023$ the instinctive primitive “`small_plan(random=weight_distance)`” will perform a similar nearest neighborhood planning algorithm in the other modules for this same navigation map (Figure 12B). However, as indicated by the argument `random=weight_distance` random fluctuations

are introduced now, so a slightly different navigation route may occur in different Navigation Module B's $n=2 \dots 1023$ (14D).

The instinctive primitive "`small_plan(random=weight_distance)`" follows a similar nearest neighbor algorithm to the one described above. However, now random fluctuations may (or may not) be introduced at each step a navigation decision is made. These fluctuations are weighted by distance position, as explained below. Normally, the destination with the shortest distance will be chosen, as seen above for Navigation Module B $n=1$ (Figure 12C). Here this is likely to occur also, but some randomness means another destination can be chosen, although the destinations the farthest away are the least likely to be chosen as the next destination, as will be shown below.

Consider that there is at any given decision point the list **destination** containing sorted destinations [a, b, c, d, e...] which still can be navigated to (124). This list is sorted by distance such that navigation to destination "a" is the shortest, then navigation to destination "b" is the next shortest, and so on (125). The value a in the list is the distance to destination "a", the value b in the list the distance to destination "b", and so on.

Consider an example where there are 5 possible destinations which the CCA7 can now navigate to from some starting point, i.e., to object "a", to object "b", to object "c", to object "d" or to object "e". As per (124) **destination** = [a, b, c, d, e], where the distance from the starting point to "a" is less than or equal to the distance from the starting point to "b", and so on (125). Object "a" represented by element a in **destination** is considered to have position=1 in the list, while object "b" has position=2, and so on. Similarly, object "a" is considered to have inverse_position=5 in this list, while object "b" has inverse_position=4, and so on (124).

Equation (126) shows that when the instinctive primitive "`small_plan(random=weight_distance)`" is triggered, a parameter "*weight*" is given a value of 4. Equation (127) shows that when the instinctive primitive "`small_plan(random=weight_distance)`" is triggered, the probability of selecting destination "x" to navigate to is given by " $\text{probability_destination}_x$ " which can be computed as " $\text{inverse_position}_x^{\text{weight}} / \sum \text{inverse_position}^{\text{weight}}$ ".

Continuing with the example above of choosing to navigation to locations "a", "b", "c", "d" or "e", consider equations (126) and (127). Consider navigating to the first destination "a" (which is the shortest navigation path from the starting point since it is the first element in **destination**). Thus, as per (127), $x = "a"$ and the value of the term " $\text{inverse_position}_a^{\text{weight}}$ " is thus 5^{weight} . The parameter $\text{weight}=4$ (126), thus the value of the term " $\text{inverse_position}_a^{\text{weight}}$ " is 5^4 , or 625. Similarly, the value of all the inverse positions raised to the fourth power ($\text{weight}=4$) added up, i.e., " $\sum \text{inverse_position}^{\text{weight}}$ " is $625+256+81+16+1=979$ (127).

(In the actual CCA7 version of the architecture, other than as needed internally (and encapsulated) for artificial neural networks being used, only very simple arithmetic is explicitly available. Thus, in (127) the " $\text{probability_destination}_x$ " is shown as being approximately equal to a term which must be calculated via high exponential powers and involves the manipulation of many decimal places. While (127) is fine for some simulations of the architecture, the relationship shown in (127) can be achieved more realistically by the architecture by making use of stored probability distributions (see below). A limited number of such probability distributions can approximate (127) in deciding which object or city to navigate next to in a planning task.)

Continuing with the example above of choosing to navigation to locations “a”, “b”, “c”, “d” or “e”, the instinctive primitive “`small_plan(random= weight_distance)`” has just been triggered. Thus, *weight* is given a value of 4 (126). The probability of the algorithm in this instinctive primitive choosing, for example, destination “a” to navigate next to, is $\text{probability_destination}_a$. By (127) this is equal to “ $= \text{inverse_position}_a^{\text{weight}} / \sum \text{inverse_position}^{\text{weight}}$ ”. Above we calculated term “ $\text{inverse_position}_a^{\text{weight}}$ ” to be 625, and the term “ $\sum \text{inverse_position}^{\text{weight}}$ ” to be 979. Thus, the probability of the algorithm in this instinctive primitive to choose destination “a” to navigate next to is 625/979, or 64%.

From similar calculations the probability of choosing any of these sorted destinations (i.e., “a” is closer, “e” is the farthest away from the starting point) in this example of [a, b, c, d, e] is [64%, 26%, 8%, 2%, .1%]. Thus, when `small_plan(random=weight_distance)` is used in this example, of the five potential destinations to choose from in [a, b, c, d, e], there is in this example, a 64% chance of navigating to the nearest neighbor “a”, a 26% chance of navigating to the next nearest neighbor “b”, but only a 0.1% chance of navigating to the farthest neighbor “e”.

In (128) it can be seen that when the instinctive primitive “`small_plan(random= False)`” is triggered, a parameter “*weight*” is given a value of 30. The result of this high *weight* is that is that the nearest neighbor destination is always used, i.e., there is little randomness (129). Thus, the probability distribution for navigation to potential objects/cities [a, b, c, d, e] is [100%, 0, 0, 0, 0], i.e., there is a 100% chance of choosing object/location “a” to navigate to, and 0% chance of choosing object/location “b”, “c”, “d” or “e” to navigate to.

As noted above, if there is only one Navigation Module B in the system, or if this is a CCA7 version of the architecture and this is Navigation Module B n=1, then as equation (118) indicates, the instinctive primitive “`small_plan(random=False)`” is triggered. The nearest neighbor (i.e., shortest distance) from the starting point of cell (0,0,0) (Figure 12C; there is a <0> put in that cell) is cell (2,2,0)—there is a <1> put tag in that cell. The next nearest neighbor is cell (3,0,0)—there is a <2> put tag in that cell. The next nearest neighbor is the only one left which is cell (4,2,0)—there is a <3> tag put in that cell. Then with no more active cells to navigate to left, there is navigation back to the starting point of (0,0,0)—there is a <0> tag there.

Once all cells are tagged, the instinctive primitive “`small_plan(random=False)`” would trigger the instinctive primitive “`move()`” to move a CCA7 robot to the tagged cells (122, 123). The instinctive primitive “`move()`” first moves the CCA7 to cell (2,2,0) with the tag <1> containing the white sphere. Then the instinctive primitive “`move()`” is triggered again to move to cell (3,0,0) with the tag <2> containing the black sphere. Then the instinctive primitive “`move()`” is triggered again to move to cell (4,2,0) with the tag <3> containing the white block. Then the instinctive primitive “`move()`” is triggered again to move to cell (0,0,0) containing the tag <0> which was the starting point. From Figure 10 and Figure 12C, note that the sum of the distances is 25+22+20+42= 109 cm in this navigation route.

Now consider the Navigation Module B’s n=2...1023 in the CCA7 version of the architecture. In Figure 12D Navigation Module B n=2 is shown. As equation (120) indicates, the instinctive primitive “`small_plan(random=False)`” is triggered. As before the instinctive primitive “`small_plan()`” considers which possible destination it can navigate to is the shortest, albeit now with a random fluctuation introduced. From the starting point of cell (0,0,0) the CCA7 can navigate

next to cells (2,2,0), (3,0,0) or (4,2,0). From Figure 10 it can be seen these correspond to possible distances of (25cm, 31cm and 42cm).

As discussed above, “`small_plan(random=weight_distance)`” will introduce a random fluctuation in deciding which object/location to navigate to via equations (124–127). The sorted list [25, 31, 42] (124, 125). The sum of the inverse positions is $3^4+2^4+1^4$, or 98, and thus the probability distribution is [81/98, 16/98, 1/98], or [83%, 16%, 1%]. The likelihood of navigating to the first position destination of 25cm corresponding to cell (2,2,0) is 83%, while the probability of navigating to (3,0,0) is 16% and the probability of navigating to (4,2,0) is 1%. A cumulative probability distribution results essentially from considering these probabilities: [$\leq 83\%$, $\leq 99\%$, $\leq 100\%$]. A random number between 0 and 1 is obtained which happens to be, for example, .55 or 55%. It is within the 83% cumulative probability corresponding to the first position destination. Thus the CCA7 architecture tags cell (2,2,0) with a $\langle 1 \rangle$.

Now the instinctive primitive “`small_plan(random=weight_distance)`” must consider navigating to the next object/location. From the starting point of cell (2,2,0) the CCA7 can navigate next to cells (3,0,0) or (4,2,0). From Figure 10 it can be seen that these correspond to possible distances of (22cm and 24cm). The sum of the inverse positions is 2^4+1^4 , or 17, and thus the probability distribution is [16/17, 1/17], or [94%, 6%]. The likelihood of navigating to the first position destination of 22cm corresponding to cell (3,0,0) is 94%, while the probability of navigating to (4,2,0) is 6%. A cumulative probability distribution results essentially from considering these probabilities: [$\leq 94\%$, $\leq 100\%$]. A random number between 0 and 1 is obtained which happens to be, in this case, for example, .95 or 95%. Thus, instead of navigating to the nearest neighbor, the CCA7 will tag the second nearest neighbor, i.e., (4,2,0) as the next destination to navigate to with a $\langle 2 \rangle$ (Figure 12D). The next nearest neighbor is the only one left which is cell (3,0,0)—there is a $\langle 3 \rangle$ tag put in that cell. Then with no more active cells to navigate to left, there is navigation back to the starting point of (0,0,0)—there is a $\langle 0 \rangle$ tag there. The tagged cells can be seen in Figure 12D.

From Figure 10 and Figure 12D note that the sum of the distances is $25+24+20+31=100$ cm in this navigation route. Thus, even though this route ended up taking a path between two locations which was not the shortest distance (i.e., going from cell (2,2,0) to cell (4,2,0) which was 24cm rather than going to cell (3,0,0) which was 22cm) it turned out that the total distance in navigating to all object/locations turned out to be shorter than the path obtained in Navigation Module B $n=1$ where the nearest neighbor algorithm was followed at each decision point.

Similar algorithms are also running in the other Navigation Module B’s at the same time. The total distance sum obtained in each Navigation Module B is transferred to the TempMap memory areas of Navigation Module B $n=0$ (119, 121). Although there are many more **TempMap** memory areas now available in the CCA7 version of the architecture, this instinctive primitive actually just keeps track of the navigation map number (i.e., which “ n ” from $n=1\dots 1023$) which has yielded the smallest total distance of the best (i.e., shortest) navigation plan found (122). Thus **Nav_ModB** _{$n=\text{best}$} where “best” is the Navigation Module B “ n ” which showed the shortest total navigation distance.

The instinctive primitive “`small_plan()`” then activates the instinctive primitive `move(WNMB’t, n=best)` (123). To continue the above example, “best” is Navigation Module B $n=2$, i.e., shown in Figure 12D. (The total distance here was 100cm versus 109cm in Navigation Module B $n=1$, and versus 119cm obtained in other Navigation Module B’s). It will then repeatedly trigger the instinctive primitive “`move()`” to navigate to $\langle 1 \rangle$ (the white sphere), then $\langle 2 \rangle$ (the white block), then $\langle 3 \rangle$ (the black sphere), and then return to the starting cell $\langle 0 \rangle$.

This navigation planning example involves navigating to three locations and then returning back to the starting position. Thus, there are only a handful of possible variations in navigation to consider, and thus, despite the random fluctuations, many variations will repeat among the over thousand Navigation Module B's. However, many real-world problems may involve more locations (or social situations or other analogous "locations") than this simple problem. In the next section, the CCA7 architecture will be applied to a larger dataset of navigation locations.

The instinctive primitive "`small_plan()`" effectively helps to decide what sequence to perform operations in. While such problems can be physically moving to different locations, they can also range from navigating in the social hierarchy space of a society to involve navigating through an idea space of more abstract concepts.

$$\mathbf{WNMB}'_{n=0\dots1023} \in \mathbb{R}^{mxnxexp} \quad (115)$$

$$\begin{aligned} &(\mathbf{WNMB}'_{n=x} \neq \text{reserved} \text{ AND } \mathbf{WNMB}'_{n=y} \neq \text{reserved}) \\ &\Rightarrow \text{initial_primitive}_{\mathbf{WNMB}'_{n=x}} = \text{initial_primitive}_{\mathbf{WNMB}'_{n=y}} \quad (116) \end{aligned}$$

$$\begin{aligned} &<\text{"go"}> \\ &\Rightarrow \mathbf{WNMA}'_t = \text{Nav_ModA.goto}() \quad (117) \end{aligned}$$

locations > 1,

$$\Rightarrow \mathbf{WNMB}'_{t, n=1} = \text{Nav_ModB}_{n=1}.\text{small_plan}(\text{random}=\text{False}) \quad (118)$$

$$\Rightarrow \mathbf{TempMap}_{\mathbf{WNMB}'_{n=0}} = \text{minimum}(\text{total_distance}_{n=1}) \quad (119)$$

$$\Rightarrow \mathbf{WNMB}'_{t, n=2\dots1023}$$

$$= \text{Nav_ModB}_{t, n=2\dots1023}.\text{small_plan}(\text{random}=\text{weight_distance}) \quad (120)$$

$$\Rightarrow \mathbf{TempMap}_{\mathbf{WNMB}'_{n=0}} = \text{minimum}(\text{total_distance}_{n=2\dots1023}) \quad (121)$$

$$\Rightarrow \text{best} = \mathbf{TempMap}_{\mathbf{WNMB}'_{n=0, \text{minimum}(\text{total_distance})}} \quad (122)$$

$$\Rightarrow \text{Nav_ModB}_{n=\text{best}}.\text{move}(\mathbf{WNMB}'_{t, n=\text{best}}) \quad (123)$$

$$\mathbf{destination} = [\mathbf{a}, \mathbf{b}, \mathbf{c}, \mathbf{d}, \mathbf{e} \dots] \quad (124)$$

$$\text{distance}_a \leq \text{distance}_b, \text{distance}_b \leq \text{distance}_c, \dots \quad (125)$$

$$\text{small_plan}(\text{random}=\text{weight_distance}),$$

$$\Rightarrow \text{weight} = 4 \quad (126)$$

$$\Rightarrow \text{probability_destination}_x \approx \text{inverse_position}_x^{\text{weight}} / \sum \text{inverse_position}^{\text{weight}} \quad (127)$$

$$\text{small_plan}(\text{random}=\text{False}),$$

$$\Rightarrow \text{weight} = 30 \quad (128)$$

$$\Rightarrow \text{probability_destination}_{a, \text{weight} > 9} = 1 \quad (129)$$

WNMB' _{n=0...1023}	<p>-Working Navigation Map B n=0 to n=1023</p> <p>-there are now 1024 Navigation Module B's, and each one has a separate Working Navigation Map B WNMB'_n</p>
WNMB' _{n=0...1023} = $\in \mathbb{R}^{minixosp}$ (115)	<p>-the current Working Navigation Map B WNMB'_{n,t} is the navigation map which the Navigation Module B_n focuses its attention on, i.e., applies operations on to make a decision to take some sort or no action</p> <p>- Navigation Module B_{n=0} to Navigation Module B_{n=1023} all operate in parallel</p> <p>- Navigation Module B_{n=0} is "reserved" for a copy of the compositional instructions, thus operations really occur in Navigation Module B's n=1 to n=1023</p>
<p>(WNMB'_x ≠ reserved AND WNMB'_y ≠ reserved)</p> <p>⇒ <i>initial_primitive</i>_{WNMB'}_{n=x} = <i>initial_primitive</i>_{WNMB'}_{n=y} (116)</p>	<p>-the same initial instinctive primitive or learned primitive is applied to all the Working Navigation Maps in the different Navigation Module B's (unless a Navigation Module B is reserved in which case any primitives will be ignored in that particular reserved Navigation Module B)</p>
<p><"go"> ⇒</p> <p>WNMA'_t = Nav_ModA.goto ()</p> <p>(117)</p>	<p>-if an instruction related to go to a place or location occurs, then the instinctive primitive goto () will be triggered</p> <p>-instinctive primitive Nav_ModA.goto () will tag in the navigation map in the Navigation Module A the cells where the CCA7 or an object is supposed to go to</p> <p>- once the instinctive primitive "goto ()" tags the cell(s) where it has to move to, it then decides if it will move (i.e., "go to") the cell with the tag; however, if there are multiple tags, i.e., multiple locations to navigate to ("locations > 1" in (118–123)), then the "small_plan ()" instinctive primitive is activated instead of moving to a single location</p>
<p>locations > 1,</p> <p>⇒ WNMB'_{t, n=1} =</p> <p>Nav_ModB_{n=1}.small_plan (random =False) (118)</p> <p>⇒ TempMap_{WNMB'}_{n=0} =</p> <p>minimum (total_distance_{n=1}) (119)</p> <p>⇒ best =</p> <p>TempMap_{WNMB'}_{n=0, minimum (total_distance)}</p> <p>(122)</p>	<p>- once activated (118), the instinctive primitive "small_plan ()" (regardless of argument) will copy the Navigation Map A to all "non-reserved" Navigation Module B's, i.e., n=1...1023 generally</p> <p>-it will remove any action words (e.g., "go" in the example above.)</p> <p>- the instinctive primitive "small_plan (random=False)" follows nearest neighbour algorithm discussed above; in making a plan where to navigate to, this primitive will choose the tag (i.e., location) that is closest to tag (i.e., location) from where it is navigating.</p> <p>-the argument random=False indicates that this instinctive primitive does not introduce any random variations</p> <p>- instinctive primitive "small_plan ()" operates on all the Navigation Module B's in parallel; the total distance sum obtained in each Navigation Module B is transferred to the TempMap memory areas of Navigation Module B n=0 (119, 121); although there are many more TempMap memory areas now</p>

	<p>available in the CCA7 version of the architecture, this instinctive primitive actually just keeps track of the navigation map number (i.e., which “n” from n=1...1023) which has yielded the smallest total distance of the best (i.e., shortest) navigation plan found (122)</p>
<p>⇒ WNMB'_{t, n=2...1023} = Nav_ModB_t n=2...1023, small_plan(random=weight_distance) (120) ⇒ TempMap_{WNMB'} n=0 = minimum(total_distance_{n=2...1023}) (121) ⇒ best = TempMap_{WNMB'} n=0, minimum(total_distance) (122)</p>	<p>- just like (118), as (120) indicates, for Navigation Module B n=2...1023 the instinctive primitive “small_plan(random=weight_distance)” will perform a similar nearest neighborhood planning algorithm in the other modules for this same navigation map (Figure 12b); however, as indicated by the argument random=weight_distance random fluctuations are introduced now, so a slightly different navigation route may occur in different Navigation Module B's n=2...1023</p> <p>- “small_plan(random=weight_distance)” follows a similar nearest neighbor algorithm to the one shown above. However, random fluctuations are introduced at each step a navigation decision is made – see below</p> <p>- the destination with the shortest distance will most likely be chosen, as seen above for Navigation Module B n=1; but some randomness means another destination can be chosen, albeit the destinations the most far away, are the least likely to be chosen as the next destination</p> <p>- instinctive primitive “small_plan()” operates on all the Navigation Module B's in parallel; the total distance sum obtained in each Navigation Module B is transferred to the TempMap memory areas of Navigation Module B n=0 (119, 121); although there are many more TempMap memory areas now available in the CCA7 version of the architecture, this instinctive primitive actually just keeps track of the navigation map number (i.e., which “n” from n=1...1023) which has yielded the smallest total distance of the best (i.e., shortest) navigation plan found (122)</p>
<p>destination = [a, b, c, d, e...] (124) distance_a ≤ distance_b, distance_b ≤ distance_c, (125)</p>	<p>- consider that there are at any given decision point the sorted destinations [a, b, c, d, e...] which still can be navigated to (124). This list is sorted by distance such that navigation to destination a is the shortest, navigation (i.e., again from the starting point) to destination b is the next shortest, and so on (125). The value a in the list is the distance to destination a, the value b in the list the distance to destination b, and so on.</p> <p>-</p>
<p>small_plan(random=weight_distance), ⇒ weight = 4 (126) ⇒ probability_destination_x ≈ inverse_position_x^{weight} / ∑ inverse_position^{weight} (127)</p>	<p>- The index value of a in [a, b, c, d, e] is 0. However, the “position” of a is considered here to be 1. The positions of these destinations a, b, c, d, e, ... can be considered to be 1, 2, 3, 4, 5, ... Consider an example where there are 5 possible destinations in total: [a, b, c, d, e]. The inversion positions of these destinations in this example, can be considered to be 5, 4, 3, 2, 1. Thus, the value of the inversion position for destination “a” would be 5. Continuing with this example and looking at equation (127), where destination x = “a”, the value of the term “inverse_position_a^{weight}” is thus 5^{weight}, or since weight=4 (126), 5⁴, or 625. Similarly, the value of all the inverse positions raised to the fourth power (weight=4) added up, i.e., “∑ inverse_position^{weight}” is 625+256+81+16+1=979 (127).</p> <p>- Equation (126) shows that when the instinctive primitive “small_plan(random= weight_distance)” is triggered, a parameter “weight” is given a value of 2. Equation (127) shows that when the instinctive primitive “small_plan(random= weight_distance)” is triggered, the probability of selecting destination “x” to navigate to is given by</p>

	<p>“probability_destination_x” which can be computed as “$\text{inverse_position}_x^{\text{weight}} / \sum \text{inverse_position}^{\text{weight}}$”.</p> <p>- Consider the example just mentioned where there are five potential destination locations which can be navigated to [a, b, c, d, e]. The instinctive primitive “small_plan(random= weight_distance)” has just been triggered. Thus, <i>weight</i> is given a value of 4 (126). The probability of the algorithm in this instinctive primitive choosing, for example, destination “a” to navigate next to, is probability_destination_a. By (127) this is approximately equal to “$\text{inverse_position}_a^{\text{weight}} / \sum \text{inverse_position}^{\text{weight}}$”. Above we calculated term “$\text{inverse_position}_a^{\text{weight}}$” to be 625, and the term “$\sum \text{inverse_position}^{\text{weight}}$” to be 979. Thus, the probability of the algorithm in this instinctive primitive to choose destination “a” to navigate next to is 625/9795, or 64%.</p> <p>-note: the exponential arithmetic of (127) is not actually performed by the CCA7 but instead the same result is obtained via the use of pre-stored probability distributions for different arrangements of destination locations</p>
<p>small_plan(random=False),</p> <p>⇒ weight = 30 (128)</p> <p>⇒ probability_destination_a, weight>9 = 1 (129)</p>	<p>-when the instinctive primitive “small_plan(random= False)” is triggered, a parameter “weight” is given a value of 30</p> <p>-the result of this high <i>weight</i> is that is that the nearest neighbor destination is always used, i.e., there is little randomness; the probability distribution for navigation to potential objects/cities [a, b, c, d, e] is [100%, 0, 0, 0, 0], i.e., there is a 100% chance of choosing object/location “a” to navigate to, and 0% chance of choosing object/location “b”, “c”, “d” or “e” to navigate to</p>
<p>⇒ best = TempMap_{WNMB'} n=0, minimum(total_distance) (122)</p> <p>⇒ Nav_ModB_{n=best}.move(WNMB'_{t, n=best}) (123)</p>	<p>- the total distance sum obtained in each Navigation Module B is transferred to the TempMap memory areas of Navigation Module B n=0 (119, 121); this instinctive primitive actually just keeps track of the navigation map number which has yielded the smallest total distance of the best (i.e., shortest) navigation plan found (122)</p> <p>- Nav_ModB_{n=best} where “best” is the Navigation Module B “n” which showed the shortest total navigation distance.</p> <p>- the instinctive primitive “small_plan()” then activates the instinctive primitive move(WNMB'_{t, n=best}) (123).</p> <p>- it will then repeatedly trigger the instinctive primitive “move()” to navigate to the tagged locations in order</p>

Table A28. Explanation of Symbols and Pseudocode in Equations (115) – (129)

Input:	<p>WNM'_t: the current Working Navigation Map A in Navigation Module A which is derived from the processed sensory inputs representing the sensory scene</p> <p>WNMB'_t: the current Working Navigation Map B in Navigation Module B n=1 which is derived from a communication sentence (a copy is also present in Navigation Module B n=0)</p>
---------------	---

	<p>WPR_i: various Working Primitives WPR's are the primitives that will be applied against the Working Navigation Map A WNM' in the Navigation Module A and/or Working Navigation Map B WNMB' in the Navigation Module B; these include:</p> <ul style="list-style-type: none"> - Nav_ModB.goto() - Nav_ModB.small_plan() - small_plan.random() (used internally by small_plan()) - Nav_ModB.minimum()) (shown for (119,121) but used internally by other primitives also) - Nav_ModB.move() <p>-Note: No advanced arithmetic is performed by any of these instinctive primitives. There is no requirement for the use of other instinctive or learned primitives for exponential arithmetic operations or complex decimal operations, as the exponential arithmetic of (127) is not actually performed by the CCA7 but instead the same result is obtained via the use of pre-stored probability distributions for different arrangements of destination locations</p>
Output:	<p><i>action_i</i>: a signal to move some actuator or send an electronic signal</p> <p>-where the CCA7 robot is physically moving or an object is being moved, the signal <i>action_i</i> is sent from Navigation Module B n=best to the Output Vector Association Module B where more detailed instructions are created to effect the required actuator outputs to the Output Vector Shaping Module to the actual real-world actuators</p>
Description:	<p>-A communication or instruction sentence is parsed in Navigation Module B and applied against a relevant represented sensory scene in Navigation Module A.</p> <p>-If the communication concerns navigating to more than one location then the instinctive primitive <i>small_plan()</i> is triggered and used to plan an optimal-like navigation route to all the specified locations, such that actual planning emerges.</p> <p>-applies to the CCA7 version of the architecture (can be retrofitted to the CCA6 version but not done at the time of writing)</p>

Table A29. Summary of Navigation Planning per Equations (115) – (129)

A.15 Large Language Module Hybridization

--To merge with CCA8 documentation --

A.16 PetitCat Robotic Device Bidirectional Interface

--To merge with CCA8 documentation --

A.17 Beta Learned Primitive System

--To merge with CCA8 documentation --

Experimental Trial Methods and Results Data

B.1 Methods

The equations (1) to (129) ~~--CCA8 to merge--~~ (above in Appendix A) are computer simulated via the Python language to represent the CCA7. The computer simulation does not interface with a real-time actual video camera or microphone inputs, or with real robotic actuators. Sensory inputs are simulated in all simulations, and actuator outputs are similarly simulated.

The navigation maps in the Python simulation have 6x6x0 dimensions (although internally a larger number of dimensions are actually used to represent segmentation of objects and binding with motion and action). As noted above navigation maps are essentially arrays. Thus, the more efficient Numpy library is called by the Python program for most operations on the navigation maps. For future larger simulations of the architecture, more classical deep learning software and hardware can be used. However, in the current simulation the FuzzyWuzzy string matching library (via pypi.org) is used for pattern matching.

The Python simulation of the architecture at this time contains a very limited set of instinctive primitives. It mainly contains the ones specified in equations (1) to (129) which relate to very basic operations and the ability for causal reasoning, analogical induction, compositionality and as discussed above in the section on new work, simple planning. At this time, instinctive primitives must be hand crafted. Automated methods for instinctive primitive creation are being explored.

Python version 3.11 is used. The parallel elements of equations (1) to (129) are simulated sequentially—a new cognitive cycle starts when all the operations of the previous cognitive cycle have completed. At present, due to the limited number of instinctive primitives and the limited storage of navigation maps representing experiences, this operation gives satisfactory run times on typical laptop or desktop personal computers (circa 2023).

The main purpose of this computer simulation is to show that the operation of the CCA7 version of the architecture is feasible, particularly its ability to perform planning operations. The simulation, i.e., based on the representation of the CCA7 version of the architecture via equations (1) to (129), can be tested below on a traveling salesperson dataset. The distances between a starting city and a dozen other cities are given in Table B1 (Google OR-Tools, 2023).

The reason for using the positional weighting in the equations (124–127) above is because the actual CCA7 version of the architecture only explicitly has access to very simple arithmetic. Thus, in weighting the probability distributions it was desired to use pre-stored distributions which could readily be accessed rather than involve complex calculations. Although (124–127) are used in the Python simulation of the architecture (albeit, necessitating the Python “Decimal” class due to the many digits created by the high-power exponents), the architecture can simply access a limited number of probability distributions based on the positions of the nearest city/location/object in a list, with no complex arithmetical calculations required.

It is possible, of course, to weight by the relative values: calculate the reciprocals of the difference of each number (nominal value of 1) in the list from the smallest value (thus, smaller difference will

give a larger reciprocal) and normalize as probabilities. For example, if there are three possible locations to navigate to with distances (arbitrary units not specified) of [12, 13, 44] then the weight probabilities would be `normalize([1/1, 1/1, 1/32])`, or [49%, 49%, 2%]. Thus, there would be a 49% chance of navigating to the location 12 units away versus a 2% chance of navigating to the location 44 units away. The probability distribution for the example above of three possible locations [12, 42, 44] is `normalize([1/1, 1/30, 1/32])`, or [94%, 3.1%, 2.9%]. In contrast, the positional probability distribution (i.e., via (124–127)) of either example yields [83%, 16%, 1%], i.e., 83% chance of navigating to the first-closest location, 16% chance of navigating to the second-closest location and a 1% chance of navigating to the third-closest location, regardless of the actual distances.

Computer simulations of the CCA7 architecture can be tested on a traveling salesperson dataset. The distances between a starting city and a dozen other cities are given in Table B1 (Google OR-Tools, 2023). This table has the distances between the following cities: 0. New York - 1. Los Angeles - 2. Chicago - 3. Minneapolis - 4. Denver - 5. Dallas - 6. Seattle - 7. Boston - 8. San Francisco - 9. St. Louis - 10. Houston - 11. Phoenix - 12. Salt Lake City

The optimal (i.e., shortest) solution obtained via brute force computation is 7293 miles. (The data in Table B1 of distances between the cities was given in miles.) The route giving this shortest path is: City #0,7,2,3,4,12,6,8,1,11,10,5,9,0 .

The traveling salesperson problem city data from Table B1 was simulated as sensory input data to the CCA7 architectures. The same CCA7 version of the architecture shown in Figure 9 was used. However, equations (126–129) were modified in different runs of the architecture as described below. As described above, the CCA7 architecture via the instinctive primitives associated with equations (126–129) attempts to produce the shortest path in a planning problem. In the case of the city data from Table B1, the architecture attempts to produce the shortest path to navigate once to the dozen cities listed in Table B1 and return to the starting city (i.e., 13 cities in total).

The following questions were asked, and the accompanying comparative experiments were then performed:

a. The position-weighted algorithm used to inject random fluctuations into the nearest neighbor algorithm (126–129) uses a *weight* parameter to create a probability distribution used in selecting the next destination to navigate to. At present a *weight* value of 4 is used.

What is the effect of varying the *weight* parameter on the shortest path yielded in the traveling salesperson problem, i.e., is the value of *weight* used in equation a reasonable one, based on a typical planning problem represented by the data in Table B1?

b. Do the multiple Navigation Module B's allow better planning in terms of this traveling salesperson problem represented by the data in Table B1?

Multiple runs to ensure statistical significance (or insignificance) of the following is to be examined: the shortest distance obtained by a CCA7 architecture modified to use only 1 Navigation Module B versus a CCA7 architecture using 1023 Navigation Module B's.

Note: When multiple Navigation Module B's are used, Navigation Module B n=0 is restricted for holding a copy of any instructions, hence, 1K-1 results in 1023 Navigation Module B's available.

Note: As per equation (118) Navigation Module B $n=1$ uses the instinctive primitive `small_plan(random=False)` thus *weight* is set to 30 for this case, i.e., nearest-neighbor algorithm without any random fluctuations. However, the Navigation Module B's $n=2 \dots 1023$ per equation (120) use the instinctive primitive `small_plan(random=weight_distance)` thus *weight* is set to 4 for these Navigation Modules and there will thus be the possibility of random fluctuations injected in choosing the next destination city at every decision point.

Note: Due to the generation of high exponents and large decimal arithmetic seen in equation (127), when the *weight* parameter exceeds 9, as per equation (129) the probability of choosing the shortest distance becomes 100%, i.e., nearest-neighbor algorithm without any random fluctuations is used.

c. Do higher quantities of Navigation Module B's result in significantly better results?

Multiple runs to ensure statistical significance (or insignificance) of the following is to be examined: the shortest distance obtained by a CCA7 architecture using 1023 (i.e., "1K") Navigation Module B's versus versions of the architecture using 4,095 (i.e., "4K") and 16,384 (i.e., "16K") Navigation Module B's.

Note: Navigation Module B $n=0$ is restricted for holding a copy of any instructions, and thus the availability of 1K-1, 4K-1 and 16K-1 Navigation Module B's, which are rounded and simply referred to as "1K", "4K" or "16K" respectively.

d. Does a "value weighted" algorithm to introduce random fluctuations (i.e., Section 4.2) give better results (i.e., a shorter distance) than the "position weighted" algorithm (i.e., equations (124) to (127))?

Multiple runs to ensure statistical significance (or insignificance) of the following is to be examined: the shortest distance obtained by the usual "position weighted" version of the CCA7 architecture using 1023 Navigation Module B's versus the shortest distance obtained by "value weighted" version of the architecture.

Note: "Position weighted" refers to equations (124) to (127) which create a probability distribution of which location to choose next in deciding where to navigate to, simply in terms of the which location is the nearest, the next-nearest, the third-next-nearest, and so on, without considering the absolute or relative values of the distances. In contrast, "valued weighted" refers to the modifications of these equations given by Section 4.2 such that the reciprocals of the difference of each distance value from the smallest value (i.e., its actual value and the actual value of the smallest distance, rather than just its position of where it is relative to the other possible destinations) are calculated and normalized to create a probability distribution of which location to choose next in deciding where to navigate to.

.

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	2451	713	1018	1631	1374	2408	213	2571	875	1420	2145	1972
1	2451	0	1745	1524	831	1240	959	2596	403	1589	1374	357	579
2	713	1745	0	355	920	803	1737	851	1858	262	940	1453	1260
3	1018	1524	355	0	700	862	1395	1123	1584	466	1056	1280	987
4	1631	831	920	700	0	663	1021	1769	949	796	879	586	371
5	1374	1240	803	862	663	0	1681	1551	1765	547	225	887	999
6	2408	959	1737	1395	1021	1681	0	2493	678	1724	1891	1114	701
7	213	2596	851	1123	1769	1551	2493	0	2699	1038	1605	2300	2099
8	2571	403	1858	1584	949	1765	678	2699	0	1744	1645	653	600
9	875	1589	262	466	796	547	1724	1038	1744	0	679	1272	1162
10	1420	1374	940	1056	879	225	1891	1605	1645	679	0	1017	1200
11	2145	357	1453	1280	586	887	1114	2300	653	1272	1017	0	504
12	1972	579	1260	987	371	999	701	2099	600	1162	1200	504	0

Table B1. Traveling Salesperson Problem. Starting at city #0, the salesperson must travel to all the other cities and then return to city #0, taking the shortest total distance. The distance between the different cities #0 to #12 are given here in miles. (Google OR-Tools, 2023)

B.2 Results Data

<i>weight value</i>	Shortest Distance (miles) Obtained from CCA7 with 1 Navigation Module B running position weighted algorithm (dataset from Table B1 used)	Shortest Distance (miles) Obtained from CCA7 with 1023 Navigation Module B's running position weighted algorithm (dataset from Table B1 used)
1	12197	8140
2	13846	7750
3	10152	7310
4	8871	7310
5	9964	7310
6	7653	7310
7	9475	7310
8	8251	7310
9	8893	7310
10	8131	8131
11	8131	8131
12	8131	8131
13	8131	8131
14	8131	8131
15	8131	8131
16	8131	8131
17	8131	8131
18	8131	8131
19	8131	8131
20	8131	8131
21	8131	8131
22	8131	8131
23	8131	8131
24	8131	8131
25	8131	8131
26	8131	8131
27	8131	8131
28	8131	8131
29	8131	8131
30	8131	8131

Table B2. Traveling Salesperson Problem. Comparison of CCA7 Architecture with 1 Navigation Module B running the position-weighted algorithm versus the architecture 1023 Navigation Module B's running the position-weighted algorithm. Single runs. Traveling Salesperson Problem data is from Table B1.

Shortest Distance (miles) Obtained from CCA7 with 1 Navigation Module B running position weighted algorithm (dataset from Table B1 used) x100 runs	Shortest Distance (miles) Obtained from CCA7 with 1 Navigation Module B running nearest-neighbor algorithm (dataset from Table B1 used) x100 runs	Shortest Distance (miles) Obtained from CCA7 with 1023 Navigation Module B's running position weighted algorithm (dataset from Table B1 used) x100 runs
number of runs: 100 mean = 9965.5 miles standard deviation = 1532.5 miles shortest path: 7647 miles % optimal (7293 miles) runs: 0%	number of runs: 100 mean = 8131.0 miles standard deviation = 0.0 miles shortest path: 8131 miles % optimal (7293 miles) runs: 0%	number of runs: 100 mean = 7432.2 miles standard deviation = 141.8 miles shortest path: 7293 miles % optimal (7293 miles) runs: 1% -versus 1 NavModB running position-weighted: $p < 0.001$ -versus 1 NavModB running nearest-neighbor: $p < 0.001$ (Welch's 1-tail t-test)
9150 9492 8120 9343 10101 9576 10547 9007 11641 7728 11050 10728 12040 10200 9213 12538 8185 11131 8614 10445 10870 8003	8131 8131	7594 7310 7310 7569 7581 7586 7653 7647 7653 7310 7310 7310 7310 7343 7293 7310 7310 7310 7310 7586 7586 7590

8279	8131	7310
9114	8131	7647
8700	8131	7310
9044	8131	7310
8812	8131	7310
8716	8131	7586
13008	8131	7310
8540	8131	7639
9520	8131	7343
12605	8131	7647
8692	8131	7310
11210	8131	7590
8131	8131	7310
8837	8131	7343
10391	8131	7310
9452	8131	7586
10934	8131	7586
12222	8131	7310
9982	8131	7310
8258	8131	7586
8665	8131	7318
8490	8131	7343
12013	8131	7586
11297	8131	7310
8785	8131	7586
7647	8131	7586
9960	8131	7310
10423	8131	7343
9615	8131	7343
11368	8131	7586
8049	8131	7310
12759	8131	7312
13601	8131	7586
12685	8131	7622
8354	8131	7586
8992	8131	7310
9243	8131	7310
9459	8131	7619
11150	8131	7320
12884	8131	7310
10963	8131	7619
10419	8131	7586
9034	8131	7343
9120	8131	7310
8895	8131	7581
8834	8131	7594

9184	8131	7310
11597	8131	7310
12538	8131	7310
9509	8131	7586
9132	8131	7310
8354	8131	7343
10902	8131	7573
8781	8131	7310
10218	8131	7310
11784	8131	7343
9003	8131	7343
9999	8131	7310
10480	8131	7619
8544	8131	7622
12939	8131	7310
8536	8131	7343
12911	8131	7320
7917	8131	7343
9053	8131	7318
8965	8131	7343
11713	8131	7310
8898	8131	7619
9773	8131	7674
8816	8131	7310
10454	8131	7310
9169	8131	7619
10435	8131	7598
11944	8131	7343
9834	8131	7586
7678	8131	7312
8884	8131	7586
13734	8131	7312

Table B3. Traveling Salesperson Problem. Comparison of CCA7 Architecture with 1 Navigation Module B running the position-weighted algorithm, with 1 Navigation Module B running the nearest-neighbor algorithm, and with 1023 Navigation Module B's running the position-weighted algorithm. Data from 100 runs of each variant of the architecture. Traveling Salesperson Problem data is from Table B1.

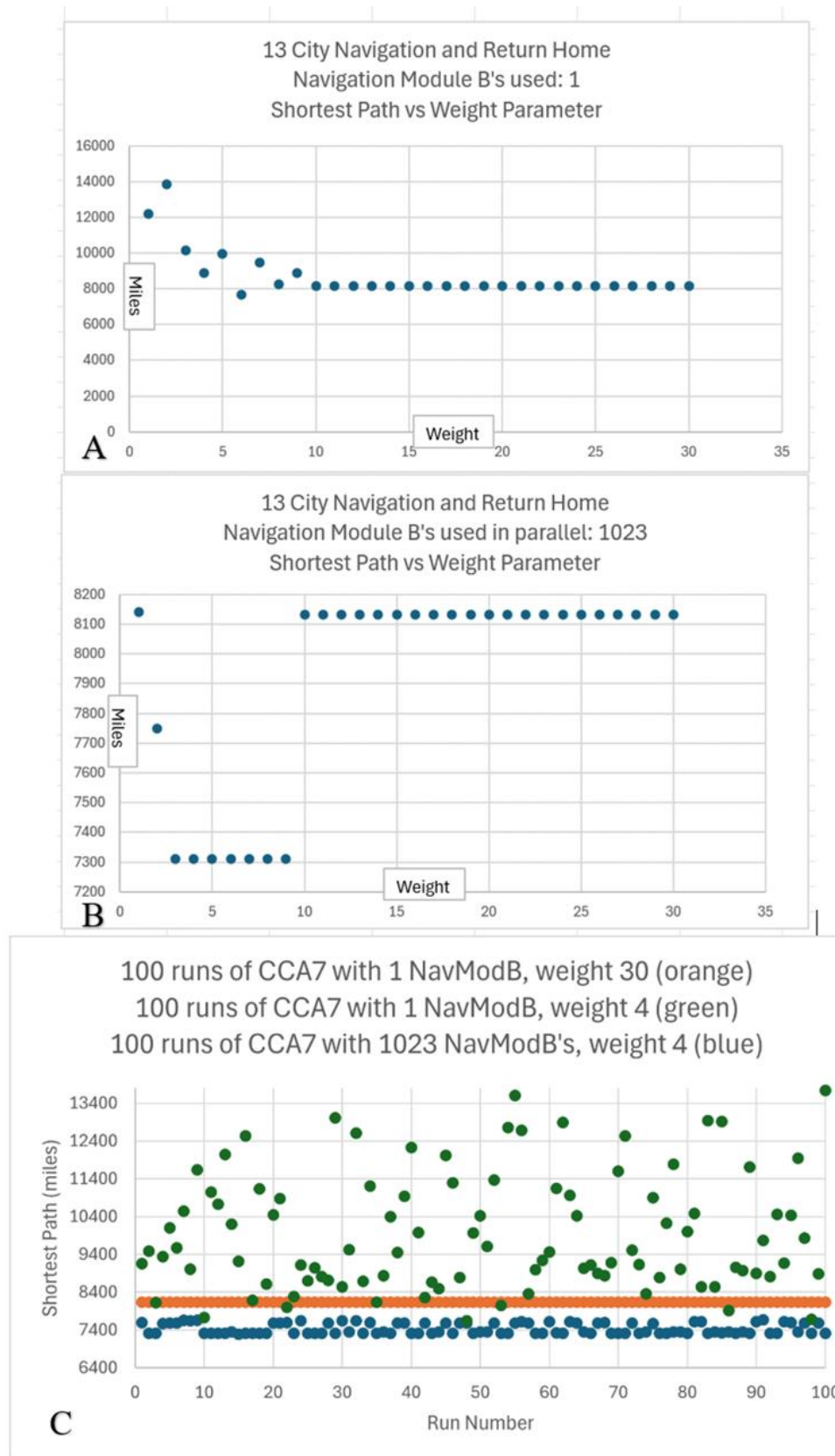


Figure B1 (Figures 13, 14 in main text). Simulation of the CCA7 robot navigating from a home city to twelve other cities (13 cities in total) and then returning home. The *weight* “wt” parameter of

equations (126, 128) is adjusted from a value of *weight* = 1 to *weight*=30 (no units). Lower values of *weight* introduce more randomness in choosing the next destination city. The shortest length (miles in the dataset used) traveled between each city and then returning home is recorded.

A. Only a single Navigation Module B is used, so rather than restrict it to equations (118, 119) (which will yield each time a value of 8131 miles per the nearest-neighbor algorithm) it is running equations (120, 121), i.e., `small_plan(random=weight_distance)` where a random fluctuation is introduced to the nearest neighbor algorithm (124–127).

B. The results are shown when the full set of Navigation Module B's $n=1\ldots 1023$ (i.e., 1023 NavModB's in parallel) are used. Here too, all modules from $n=1$ to $n=1023$ were allowed to follow equations (120, 121), i.e., `small_plan(random=weight_distance)` where a random fluctuation is introduced to the nearest neighbor algorithm. At higher values of the *weight* parameter, the CCA7 follows the nearest neighbor algorithm in deciding which city to navigate to next—a value of 8131 miles is expected for the data set used. At lower values of the *weight* parameter, there are more random fluctuations introduced in choosing the next city to navigate to (equations (124–129)). Note that in **A** where only a single Navigation Module B is used, the extra randomness generally did not help to yield a shorter path than the nearest neighbor algorithm. Note that in **B** where 1023 Navigation Module B's were used in parallel, the randomness provided within a range of the *weight* parameters, helped to yield shorter paths than obtained via the nearest neighbor algorithm, approaching the theoretical shortest path of 7293 miles.

C. 100 simulated runs with the same dataset as above when there is only one Navigation Module B used, with a weight of 30, i.e., `small_plan(random=weight_distance)`, where there is only one Navigation Module B used, with a weight of 4, i.e., `small_plan(random=weight_distance)`, and where there are 1023 Navigation Module B's used, with a weight of 4, i.e., `small_plan(random=weight_distance)`. Note that when there is only one Navigation Module B used, the nearest neighbor algorithm without any random fluctuations gives the best results. However, if the full 1023 Navigation Module B's are used, the random fluctuations are advantageous in yielding shorter paths.

Shortest Distance (miles) Obtained from CCA7 with 1023 (“1K”) Navigation Module B’s running position weighted algorithm (dataset from Table B1 used) x100 runs	Shortest Distance (miles) Obtained from CCA7 with 4K Navigation Module B’s running position weighted algorithm (dataset from Table B1 used) x100 runs	Shortest Distance (miles) Obtained from CCA7 with 16K Navigation Module B’s running position weighted algorithm (dataset from Table B1 used) x100 runs
number of runs: 100 mean = 7432.2 miles standard deviation = 141.8 miles shortest path: 7293 miles % optimal (7293 miles) runs: 1%	number of runs: 100 mean = 7309.1 miles standard deviation = 29.8 miles shortest path: 7293 miles % optimal (7293 miles) runs: 27% -versus 1K NavModB’s p < 0.001 (Welch’s 1-tail t-test)	number of runs: 100 mean = 7296.8 miles standard deviation = 6.9 miles shortest path: 7293 miles % optimal (7293 miles) runs: 67% -versus 4K NavModB’s p < 0.001 (Welch’s 1-tail t-test)
7594 7310 7310 7569 7581 7586 7653 7647 7653 7310 7310 7310 7310 7343 7293 7310 7310 7310 7310 7586 7586 7590 7310 7647 7310	7310 7310 7295 7320 7293 7310 7310 7310 7310 7310 7293 7310 7310 7293 7293 7310 7310 7310 7295 7310 7310 7293 7310 7310	7310 7293 7293 7293 7293 7293 7310 7293 7310 7293 7310 7293 7293 7293 7295 7293 7293 7293 7295 7293 7293 7293 7293 7295

7310	7310	7295
7310	7310	7310
7586	7310	7293
7310	7293	7293
7639	7310	7293
7343	7310	7310
7647	7310	7310
7310	7310	7293
7590	7293	7293
7310	7310	7310
7343	7293	7293
7310	7310	7293
7586	7293	7293
7586	7310	7310
7310	7293	7293
7310	7293	7293
7586	7293	7310
7318	7310	7293
7343	7293	7293
7586	7310	7293
7310	7310	7293
7586	7310	7295
7586	7310	7295
7310	7310	7293
7343	7310	7293
7343	7310	7293
7586	7310	7310
7310	7320	7310
7312	7310	7293
7586	7293	7310
7622	7293	7293
7586	7293	7310
7310	7310	7293
7310	7293	7293
7619	7295	7295
7320	7310	7293
7310	7312	7293
7619	7343	7310
7586	7310	7293
7343	7310	7293
7310	7293	7293
7581	7293	7310
7594	7310	7295
7310	7312	7293
7310	7312	7310
7310	7293	7293

7586	7586	7293
7310	7310	7295
7343	7310	7293
7573	7310	7293
7310	7310	7293
7310	7343	7293
7343	7310	7293
7343	7293	7310
7310	7310	7293
7619	7310	7293
7622	7293	7293
7310	7310	7310
7343	7293	7295
7320	7310	7293
7343	7293	7310
7318	7318	7293
7343	7310	7295
7310	7310	7310
7619	7310	7293
7674	7310	7293
7310	7310	7293
7310	7310	7293
7619	7293	7293
7598	7293	7293
7343	7293	7295
7586	7343	7293
7312	7310	7293
7586	7293	7293
7312	7310	7293

Table B4. Traveling Salesperson Problem. Comparison of CCA7 Architectures all running the position-weighted algorithm but with 1K (actually 1K-1, or, 1023) Navigation Module B's, with 4K Navigation Module B's and with 16K Navigation Module B's. Data from 100 runs of each variant of the architecture. Traveling Salesperson Problem data is from Table B1.

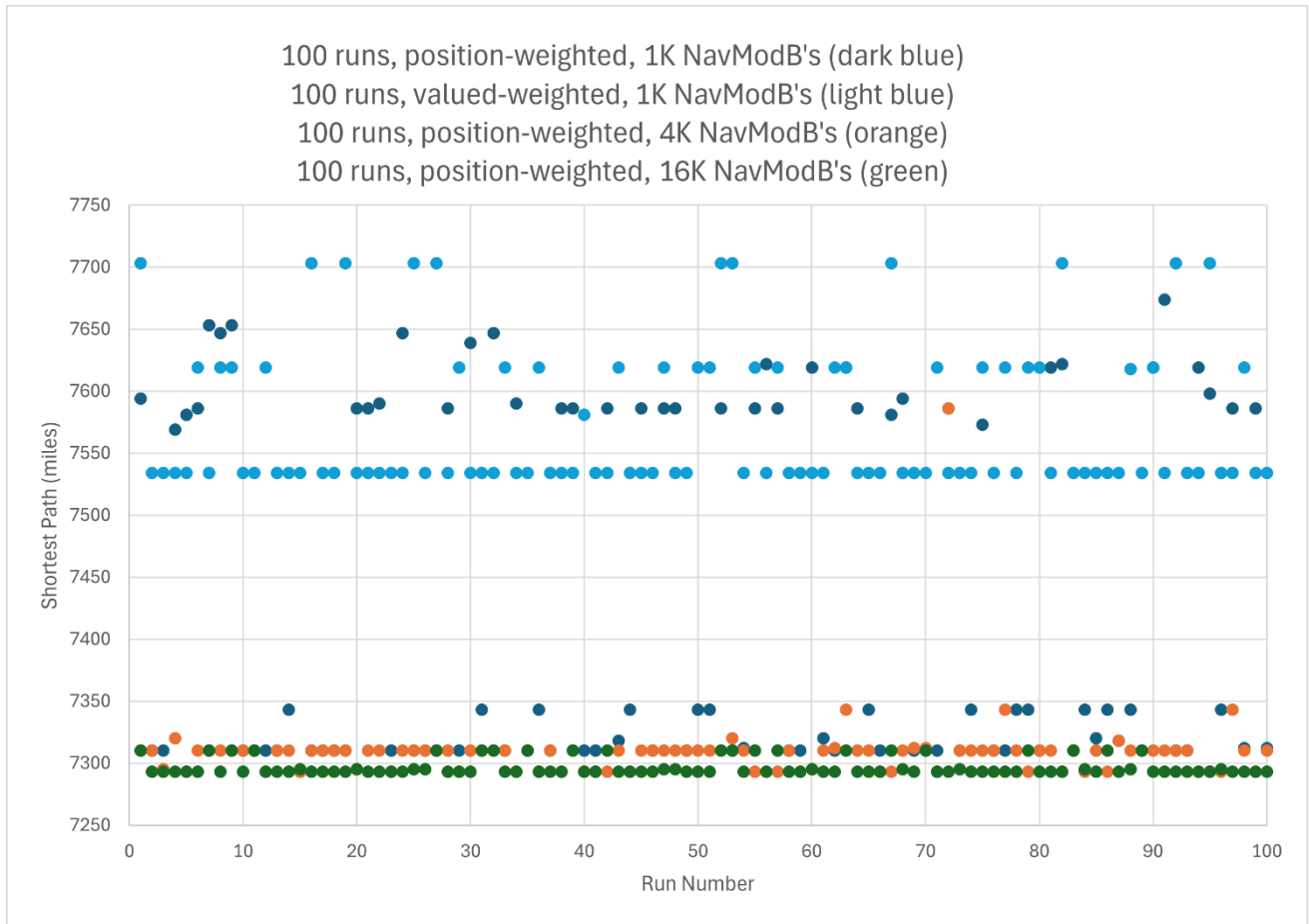


Figure B2 (Figure 15 in main text). Traveling Salesperson Problem. Comparison of three CCA7 Architectures running the position-weighted algorithm but with 1K Navigation Module B's (dark blue dots), with 4K Navigation Module B's (orange dots), and with 16K Navigation Module B's (green dots). There is also comparison with the architecture with 1K Navigation Module B's but running a value-weighted algorithm (light blue dots). Data from 100 runs of each variant of the architecture. Traveling Salesperson Problem data is from Table B1.

Shortest Distance (miles) Obtained from CCA7 with 1023 Navigation Module B's running position weighted algorithm (dataset from Table B1 used) x100 runs	Shortest Distance (miles) Obtained from CCA7 with 1023 Navigation Module B's running value-weighted algorithm (dataset from Table B1 used) x100 runs
number of runs: 100 mean = 7432.2 miles standard deviation = 141.8 miles shortest path: 7293 miles % optimal (7293 miles) runs: 1% -versus 1 NavModB running value-weighted: p < 0.001 (Welch's 1-tail t-test)	number of runs: 100 mean = 7572.6 miles standard deviation = 58.0 miles shortest path: 7534 miles % optimal (7293 miles) runs: 0%
7594 7310 7310 7569 7581 7586 7653 7647 7653 7310 7310 7310 7310 7343 7293 7310 7310 7310 7310 7586 7586 7590 7310	7703 7534 7534 7534 7534 7619 7534 7619 7619 7534 7534 7619 7534 7534 7534 7703 7534 7534 7703 7534 7534 7534 7534

7647	7534
7310	7703
7310	7534
7310	7703
7586	7534
7310	7619
7639	7534
7343	7534
7647	7534
7310	7619
7590	7534
7310	7534
7343	7619
7310	7534
7586	7534
7586	7534
7310	7581
7310	7534
7586	7534
7318	7619
7343	7534
7586	7534
7310	7534
7586	7619
7586	7534
7310	7534
7343	7619
7343	7619
7586	7703
7310	7703
7312	7534
7586	7619
7622	7534
7586	7619
7310	7534
7310	7534
7619	7534
7320	7534
7310	7619
7619	7619
7586	7534
7343	7534
7310	7534
7581	7703
7594	7534
7310	7534

7310	7534
7310	7619
7586	7534
7310	7534
7343	7534
7573	7619
7310	7534
7310	7619
7343	7534
7343	7619
7310	7619
7619	7534
7622	7703
7310	7534
7343	7534
7320	7534
7343	7534
7318	7534
7343	7618
7310	7534
7619	7619
7674	7534
7310	7703
7310	7534
7619	7534
7598	7703
7343	7534
7586	7534
7312	7619
7586	7534
7312	7534

Table B5. Traveling Salesperson Problem. Comparison of two CCA7 Architectures each with 1023 (i.e., “1K”) Navigation Module B’s, with one variant running the position-weighted algorithm and the other variant running the value-weighted algorithm. Data from 100 runs of each variant of the architecture. Traveling Salesperson Problem data is from Table B1.

--to merge with experimental data AGI2024 experiments--

B.3 Additional Methods and Results Data

Compositionality prompt for GPT4

“There is a black sphere and to the right a black cube and to the right another black cube, and behind the middle black cube a white cylinder. If you place the black sphere on top of the black cube which is not near a cylinder then tell me what the scene looks like now.”

TSP prompt for GPT4

Starting at city #0 you need to travel to each of 12 other cities once, and then return back to city#0. Below is data matrix of the distances from #0 to the other 12 cities, from city #1 to the other 12 cities, and so on. Please give a shortest distance.

[0, 2451, 713, 1018, 1631, 1374, 2408, 213, 2571, 875, 1420, 2145, 1972],
[2451, 0, 1745, 1524, 831, 1240, 959, 2596, 403, 1589, 1374, 357, 579],
[713, 1745, 0, 355, 920, 803, 1737, 851, 1858, 262, 940, 1453, 1260],
[1018, 1524, 355, 0, 700, 862, 1395, 1123, 1584, 466, 1056, 1280, 987],
[1631, 831, 920, 700, 0, 663, 1021, 1769, 949, 796, 879, 586, 371],
[1374, 1240, 803, 862, 663, 0, 1681, 1551, 1765, 547, 225, 887, 999],
[2408, 959, 1737, 1395, 1021, 1681, 0, 2493, 678, 1724, 1891, 1114, 701],
[213, 2596, 851, 1123, 1769, 1551, 2493, 0, 2699, 1038, 1605, 2300, 2099],
[2571, 403, 1858, 1584, 949, 1765, 678, 2699, 0, 1744, 1645, 653, 600],
[875, 1589, 262, 466, 796, 547, 1724, 1038, 1744, 0, 679, 1272, 1162],
[1420, 1374, 940, 1056, 879, 225, 1891, 1605, 1645, 679, 0, 1017, 1200],
[2145, 357, 1453, 1280, 586, 887, 1114, 2300, 653, 1272, 1017, 0, 504],
[1972, 579, 1260, 987, 371, 999, 701, 2099, 600, 1162, 1200, 504, 0]

GPT-3.5 API Results

Response from OpenAI:

The scene now looks like a black sphere on top of a black cube, with another black cube to the right and a white cylinder behind it.

Response from OpenAI:

The scene now looks like a black sphere sitting on top of a black cube, with another black cube to the right and a white cylinder behind it. The contrast between the black and white objects creates a stark and geometric aesthetic.

Response from OpenAI:

The scene now looks like a stack of objects, with the black sphere on top of the black cube. The two black cubes are still on the right side, with the white cylinder behind the middle black cube. The black sphere may or may not be perfectly balanced on top of the black cube, depending on their sizes and positions. The overall scene still has a predominantly black color scheme, with the white cylinder providing a contrast.

Response from OpenAI:

The scene now looks like a black sphere on top of a black cube, with another black cube to the right and a white cylinder behind the middle black cube.

Response from OpenAI:

The scene now looks like a black sphere on top of a black cube, with another black cube to the right and a white cylinder behind the middle black cube.

Response from OpenAI:

The scene now looks like a stack of three black objects, with the black sphere on top of the black cube and the other black cube to the right. The white cylinder is still behind the middle black cube.

Response from OpenAI:

The scene now consists of a black sphere resting on top of a black cube, with another black cube to the right and a white cylinder behind it. The black sphere may partially obscure the black cube it is resting on, and the white cylinder may be partially visible behind it. Overall, the scene appears to have a sense of balance and symmetry, with the black objects forming a triangle shape.

Response from OpenAI:

The scene now looks like a black sphere sitting on top of a black cube, with another black cube to the right. Behind the middle black cube, there is a white cylinder. The rest of the scene remains the same.

Response from OpenAI:

The scene now looks like a black sphere sitting on top of a black cube, with another black cube on the right. Behind the middle black cube is a white cylinder.

Response from OpenAI:

Now, the scene would look like a black sphere sitting on top of a black cube, with another black cube next to it and a white cylinder behind it. The overall composition would have a more balanced and symmetrical appearance.

Response from OpenAI:

The scene now appears as a stack of objects, with the black sphere placed on top of the black cube. Behind the stack, the white cylinder can still be seen. The two black cubes are now aligned horizontally, with the black sphere acting as a bridge between them.

Response from OpenAI:

The scene now looks like a black sphere on top of a black cube, with another black cube to the right and a white cylinder behind the middle black cube.

Response from OpenAI:

The scene now looks like a black sphere sitting on top of a black cube, with another black cube to the right and a white cylinder behind it. The black sphere appears to be balanced on top of the cube, and the cylinder is partially hidden behind the cube. The overall color scheme is still mostly black, with a pop of white from the cylinder.

Response from OpenAI:

The scene now looks like a tower, with the black sphere sitting on top of the black cube and the two black cubes stacked on top of each other. The white cylinder is still visible behind the middle black cube. The overall color scheme is still mostly black, with a pop of white from the cylinder.

Response from OpenAI:

The scene now shows a black sphere on top of a black cube, with another black cube to the right. Behind the middle black cube is a white cylinder.

Response from OpenAI:

Now the scene looks like there is a black sphere sitting on top of a black cube, with another black cube to the right of it. Behind the middle black cube, there is a white cylinder. The black sphere stands out against the dark background of the cubes, creating a contrast. The white cylinder adds a touch of brightness to the scene.

Response from OpenAI:

The scene now has a black sphere resting on top of a black cube, which is surrounded by another black cube on the right. Behind the middle black cube, there is a white cylinder. The black sphere stands out against the black and white background, creating a contrast in color and shape.

Response from OpenAI:

The scene now looks like a black sphere sitting on top of a black cube, with another black cube to the right. Behind the middle black cube is a white cylinder. The black sphere is the tallest object in the scene, with the two black cubes and the white cylinder forming a row behind it. The overall color scheme is predominantly black, with a pop of white from the cylinder.

Response from OpenAI:

The scene now looks like a black sphere on top of a black cube, with another black cube and a white cylinder in the background. The arrangement of objects has not changed, but the black sphere is now elevated on top of the cube.

Response from OpenAI:

The scene now looks like a black sphere sitting on top of a black cube, with another black cube on its right. Behind the middle black cube is a white cylinder, which is still visible. The overall scene still has a dark color scheme, with the contrast between the black objects and the white cylinder adding visual interest.

C:\Users\howar>

GPT-4 Results



You

there is a black sphere and to the right a black cube and to the right another black cube, and behind the middle black cube a white cylinder. If you place the black sphere on top of the black cube which is not near a cylinder then tell me what the scene looks like now.

ChatGPT

ChatGPT

With the black sphere now placed on top of the black cube that isn't near the cylinder, the scene looks like this:

- The black sphere is resting on top of one of the black cubes.
- The other black cube remains to the right of the cube with the sphere on it.
- Behind the middle cube (the one with the sphere on top or the one to its right, depending on their original positions), there's a white cylinder.

So, you have a black cube with a black sphere on top of it, next to another black cube, and behind one of these cubes, there's a white cylinder.

ChatGPT

ChatGPT

With the black sphere now placed on top of the black cube that is furthest from the white cylinder, the scene has the following configuration:

- To the left, there is a stack with a black sphere on top of a black cube.
- In the middle, there remains a single black cube.
- Behind this middle black cube, there is a white cylinder.

- To the right of the middle black cube (which is near the white cylinder), there is no object since the previously mentioned black cube is now part of the stack on the left.

This adjustment creates a more vertically oriented structure on one side of the scene while maintaining a clear separation between the objects, with the white cylinder still positioned behind the middle object but now only near one black cube.

ChatGPT

ChatGPT

To understand the scene after placing the black sphere on top of the black cube that is not near a cylinder, let's break down the initial setup and the change requested:

1. Initial Setup:

- A black sphere.
- To the right of the black sphere, a black cube (let's call it Cube 1).
- To the right of Cube 1, another black cube (Cube 2).
- Behind Cube 1 (and not Cube 2, based on your instructions), there's a white cylinder.

2. Change Requested:

- Place the black sphere on top of the black cube which is not near a cylinder.

Given the description, Cube 2 is the one that is not near the cylinder, as the cylinder is specified to be behind Cube 1. Thus, the sphere should be placed on top of Cube 2.

3. Resulting Scene:

- Cube 1 remains in its original position.
- The white cylinder remains behind Cube 1.
- Cube 2 now has the black sphere on top of it.

This arrangement leads to a scene with two distinct groups: the first group consisting of Cube 1 with the white cylinder behind it, and the second group consisting of Cube 2 with the black sphere on top. Cube 1 and the cylinder are isolated from Cube 2 and the sphere by the spatial arrangement described.

3 / 3

In the updated scene, the black sphere is placed on top of the rightmost black cube (since it's specified that the cube should not be near a cylinder). The arrangement from left to right would then be: a black cube with a white cylinder behind it, some space between them, and another black cube with a black sphere on top of it.

ChatGPT

With the given setup, we start with three objects aligned from left to right: a black sphere, a black cube (with a white cylinder behind it), and another black cube. If you place the black sphere on top of the black cube that is not near the cylinder, it means you would move the sphere from its original position to the top of the rightmost black cube.

Now, the scene would look like this:

- On the left, there's an empty space where the black sphere originally was.
- In the middle, there's a black cube with a white cylinder positioned directly behind it.
- On the right, there's a black cube with the black sphere now on top of it.

This reconfiguration creates a more vertically structured element on the right side, adding a layer of complexity to the arrangement with the inclusion of height differentiation.

5 / 5

If you place the black sphere on top of the black cube which is not near the cylinder, this changes the scene as follows:

- The leftmost black cube now has the black sphere on top of it, forming a sort of totem with a sphere-cube structure.
- The rightmost black cube remains unchanged and is still just a cube on its own.
- The white cylinder remains positioned behind the middle cube, unaffected by the repositioning of the black sphere.

So, the new scene from left to right is: a black sphere on top of a black cube, a black cube with a white cylinder behind it, and another black cube to the far right.

Placing the black sphere on top of the black cube that is not near a cylinder creates a scene with two primary structures. On the left, there is a composition consisting of a black sphere resting on top of a black cube. This forms a vertically stacked arrangement. To the right of this structure, there is a standalone black cube. Behind this standalone black cube, slightly offset to the background, is a white cylinder. The overall scene presents a contrast in shapes and colors, with the geometric forms of the cubes and sphere in black sharply contrasting against the cylindrical shape and white color of the cylinder in the background.

In the new configuration, you have a black sphere placed on top of the leftmost black cube, since the original scene described a black cube to the right of the sphere, another black cube to the right of the first cube, and a white cylinder behind the middle black cube. This means the leftmost cube is the one not near the cylinder. So, after placing the black sphere on top of this cube, the scene would consist of:

- A black sphere on top of a black cube on the left side.
- To the right of this stacked pair, there is another black cube with a white cylinder behind it.

This setup gives you a visually interesting arrangement with the black sphere elevated on the cube, drawing attention to the left side of the scene while maintaining a balance with the standalone black cube and the white cylinder on the right.

To visualize your described setup after placing the black sphere on top of the specified black cube, let's break down the initial configuration and the change you described:

- Initially, you have a black sphere, a black cube to the right of the sphere, another black cube further to the right, and a white cylinder behind the middle black cube.
- You want the black sphere placed on top of the black cube that is not near a white cylinder. This means the sphere should be placed on top of the far-right black cube, as the middle cube is near the cylinder.

After making the change, the scene looks like this:

- The first (middle) black cube remains in its original position, with the white cylinder directly behind it.
- The far-right black cube now has the black sphere on top of it.

The visual arrangement from left to right now is: a black cube with a white cylinder behind it, and further to the right, another black cube with a black sphere on top of it.

If we follow the instructions given, the scene starts with three objects arranged in a line from left to right: a black sphere, a black cube (with a white cylinder behind it), and another black cube. You've asked to place the black sphere on top of the black cube that is not near the cylinder.

This means the sphere should be placed on top of the rightmost black cube (since the middle black cube has the white cylinder behind it, and the instructions specify the cube not near the cylinder).

So, the modified scene would now have, from left to right, a black cube with a white cylinder behind it, and another black cube with a black sphere on top of it.

If you place the black sphere on top of the black cube which is not near a cylinder, then the scene would look like this:

1. There is a stack composed of a black cube at the bottom and a black sphere on top of it.
2. To the right of this stack, there is a black cube.
3. Behind this second black cube, there is a white cylinder.

So, from left to right, you first see the stack of the black sphere on the black cube, then the second black cube, with the white cylinder slightly hidden behind the second black cube.

To visualize the updated scene after placing the black sphere on top of the black cube that is not near the white cylinder, let's break down the initial setup and the change:

1. Initially, you have a black sphere.
2. To the right of the black sphere, there's a black cube.
3. To the right of this first black cube, there's another black cube.
4. Behind the middle black cube (which would be the first black cube), there's a white cylinder.

The instruction is to place the black sphere on top of the black cube that is not near the white cylinder. Given the setup, this means placing the black sphere on top of the second black cube, since the first black cube is the one near the white cylinder.

After this change, the scene from left to right would look like this:

- The first black cube (with the white cylinder behind it).
- The second black cube with the black sphere on top of it.

Now, let's visualize this updated arrangement.

Now, let's visualize this updated arrangement.

Now, let's visualize this updated arrangement.



The updated scene has been visualized with the black sphere placed on top of the second black cube, which is not near the white cylinder. The first black cube remains with the white cylinder behind it, and the arrangement from left to right follows your instructions.

ChatGPT

ChatGPT

With your adjustments, the scene now consists of, from left to right:

1. A black cube with a black sphere on top.
2. A black cube.
3. Behind the second black cube, there's a white cylinder.

So, you have a stacked structure on the left, a solitary cube in the middle, and a cylinder placed behind the middle cube, making for a visually tiered arrangement with varying shapes and a clear contrast between the objects and their arrangement.

ChatGPT

ChatGPT

In the scene you described, there are initially three black objects: a sphere, and two cubes, with the cubes being to the right of the sphere, and a white cylinder positioned behind the middle black cube. If you place the black sphere on top of the black cube that is not near the white cylinder, this means you'll be stacking the sphere on the farthest black cube to the right.

So, after this action, from left to right, you would have a black cube with nothing on top of it, then a black cube with a black sphere on top, and behind this second cube (but likely obscured from some angles), you would have a white cylinder. The scene would consist of these four objects, with one of the black cubes now serving as a pedestal for the black sphere.

In the scene you've described, there are three black objects (a sphere and two cubes) and one white cylinder. Initially, the objects are arranged with the sphere and two cubes in a line, and the cylinder is behind the middle cube. If you place the black sphere on top of the black cube that is not near the cylinder, the scene changes as follows:

- The black sphere is now on top of one of the black cubes.
- Since the cylinder is behind the middle cube, the cube that is not near the cylinder would be the cube on one of the ends. This means the sphere is now placed on top of either the first or the last cube in the line, depending on which one is further from the cylinder.
- The white cylinder remains behind the middle cube, unchanged.

So, if we consider the arrangement from left to right as sphere, cube, cube (with the cylinder behind the middle cube), after the adjustment, the sphere would be on top of the cube on the end farthest from the cylinder, making a vertical arrangement of sphere atop cube. The cylinder still remains behind the middle cube. This creates a visually layered scene with distinct geometric shapes in black and white.

ChatGPT

With the adjustments made to the scene, here's what it would look like now:

- To the far left, there's a black cube with nothing on top of it or near it. This cube is isolated from the rest of the objects.
- Moving right from the isolated black cube, there's a black sphere. Originally, the black sphere was on the far left, but now it's been placed on top of the black cube that is near the white cylinder. This forms a stack: the black cube at the bottom, the black sphere on top of this cube, and the white cylinder stands behind this particular stack.
- The white cylinder remains in its original position, which is behind the middle object in the scene (now the stack of the black cube and black sphere).
- To the right of the scene, there's another black cube, which is the one that originally had the black sphere placed on top of it according to your instruction. This cube is now to the right of the cylinder and has nothing on top of it, as the sphere has been moved.

This arrangement creates a visually interesting scene with a mix of shapes and contrasting colors, positioning the objects in a manner that provides depth and layers to the setup.

To understand the scene you've described and the change you want to make, let's break it down:

1. There's a black sphere.
2. To the right of the black sphere, there's a black cube (Cube 1).
3. To the right of this first black cube, there's another black cube (Cube 2).
4. Behind the middle black cube (Cube 1), there's a white cylinder.

So, before any changes, from left to right, the arrangement is: black sphere, black cube (Cube 1) with a white cylinder behind it, and another black cube (Cube 2).

You've asked to place the black sphere on top of the black cube which is not near a cylinder. That means the sphere should be placed on top of Cube 2, since Cube 1 is the one near the cylinder.

After making this change, the scene looks like this:

- From left to right, there's a black cube (Cube 1) with a white cylinder behind it.
- To the right of Cube 1, there's another black cube (Cube 2) with the black sphere now on top of it.

This creates a more layered setup with Cube 1 still aligned with the white cylinder behind it, and Cube 2 now acting as a pedestal for the black sphere.

ChatGPT

ChatGPT

If we follow your instructions, we initially have three objects aligned left to right: a black sphere, a black cube (with a white cylinder behind it), and another black cube. You asked to place the black sphere on top of the black cube that is not near the cylinder. This means the sphere should be placed on top of the far-right black cube since it's the one not near the cylinder.

Now, the scene would look like this: to the far left, there's a black cube with a white cylinder directly behind it. To the right of this setup, there's another black cube, but this one has the black sphere on top of it, creating a sort of tower. This configuration keeps the objects spatially organized in a manner that follows your instructions, altering the scene to accommodate the sphere's new position atop the far-right cube.

In the initial scene, there are three objects arranged from left to right: a black sphere, a black cube (with a white cylinder behind it), and another black cube. The instruction is to place the black sphere on top of the black cube that is not near the cylinder.

This means taking the black sphere and placing it on top of the third object, the black cube on the far right, since it's the one not near the white cylinder.

After this modification, the scene would look like this, from left to right:

1. A black cube with a white cylinder directly behind it.
2. Empty space where the black sphere originally was.
3. A black cube with the black sphere now on top of it.

This arrangement results in a vertical stack of the black sphere on the rightmost black cube, while the middle black cube still has the white cylinder positioned behind it.

TSP Results GPT3 and GPT4

8464
8970
8154
10118
9392
10804
11158
13729
10527
13047
7379
9221
8737
12275
10903
8391
12042
9594
10994
10527

7594
7310
7310
7569
7581
7586
7653
7647
7653
7310
7310
7310
7310
7343
7293

7310
7310
7310
7310
7586