PROYECTO DE GRADO

Presentado ante la ilustre UNIVERSIDAD DE LOS ANDES como requisito parcial para obtener el Título de INGENIERO DE SISTEMAS

BRAINCEMISID 4.0: DISEÑO E IMPLEMENTACIÓN EN PYTHON DEL MOVIMIENTO AUTÓNOMO. MÓDULO C: HISTORIA Y EPISODIO DE VIDA

Por

Br. Daniel Asdrubal Leo Daza
Tutor: Dr. Gerard Páez Monzón

Agosto 2023

BRAINCEMISID 4.0: DISEÑO E IMPLEMENTACIÓN EN PYTHON DEL MOVIMIENTO AUTÓNOMO. MÓDULO C: HISTORIA Y EPISODIO DE VIDA

Br. Daniel Asdrubal Leo Daza

Proyecto de Grado — Sistemas Computacionales, 60 páginas

Resumen: En este proyecto de grado, se aborda el desarrollo de una memoria artificial en el contexto de un sistema de movimiento autónomo. El objetivo principal del proyecto es diseñar e implementar el módulo C, enfocado en la gestión de la historia y el episodio de vida, dentro de la plataforma BrainCEMISID 4.0. Se emplea el lenguaje de programación Python para la implementación de la memoria artificial y se utilizan metodologías ágiles y adaptativas en el proceso de desarrollo. El proyecto busca emular las funciones cognitivas de la memoria humana, incorporando elementos emocionales, biológicos y culturales en la representación de la historia de vida.

Palabras clave: Memoria artificial, Python, movimiento autónomo, integración de módulos, sistema cognitivo.

Índice

Índice de figuras

Capítulo 1
Introducción

En la era actual de la inteligencia artificial, la relevancia de la memoria en las máquinas inteligentes se ha vuelto cada vez más evidente. La memoria artificial juega un papel crucial, permitiendo a estos sistemas adquirir conocimiento, aprender de experiencias pasadas y tomar decisiones informadas de manera similar al cerebro humano.

El objetivo principal de este proyecto de grado es abordar la complejidad inherente a la integración y la interacción eficiente de la memoria artificial con otros módulos especializados, como el módulo de atención y la red sensorial que procesa eventos. Mediante la superación de estos desafíos, se busca que la máquina inteligente en estudio exhiba un nivel más avanzado de inteligencia y comportamiento autónomo.

En este proyecto, se explorará cómo mejorar la interconexión de estos módulos relevantes para fortalecer la memoria artificial en el desarrollo de máquinas inteligentes. El propósito es alcanzar avances significativos en áreas como el reconocimiento de patrones y la toma de decisiones, permitiendo así un mayor rendimiento y aplicaciones más amplias de la inteligencia artificial en diversos contextos cotidianos.

1.1 Planteamiento del problema

En el campo de la inteligencia artificial, se plantea un desafío fascinante y complejo: ¿Cómo podemos emular la asombrosa capacidad de memoria del cerebro humano en máquinas inteligentes? El

cerebro humano destaca por su habilidad para almacenar, organizar y recuperar información de manera eficiente y adaptable, permitiéndonos aprender, recordar y tomar decisiones informadas.

Sin embargo, para abordar este desafío, es crucial no solo enfocarse en el desarrollo de una memoria artificial, sino también en la complejidad de integrarla efectivamente con otros dos módulos fundamentales: el módulo de atención y una red sensorial que captura y procesa eventos dinámicos.

La investigación se concentrará en identificar patrones y estrategias utilizadas por el cerebro humano para la codificación y decodificación de la información, buscando aplicar estos conocimientos en el diseño de la memoria artificial.

Además, se abordará la complejidad de integrar efectivamente la memoria con los módulos de atención y la red sensorial. Estos módulos desempeñan un papel crucial en la selección y focalización de información relevante, así como en la captura y procesamiento de eventos dinámicos del entorno. La sincronización adecuada entre estos módulos resulta esencial para lograr una colaboración sinérgica y potenciar el rendimiento de las máquinas inteligentes.

1.2 Justificación

La integración de una memoria artificial con un módulo de atención y un módulo de una red sensorial que procesa eventos discretos presenta una oportunidad única para mejorar la eficacia y la adaptabilidad de las máquinas inteligentes. Aunque existen máquinas inteligentes con memorias artificiales más avanzados en el campo (Hassabis, D., Maguire, E. A, 2009 y Taewoon Kim, Michael Cochez, Vincent Francois-Lavet, Mark Neerincx, Piek Vossen 2022), este enfoque particular busca abordar las limitaciones asociadas con la atención y el procesamiento de eventos discretos. La capacidad de atención permitirá una selección más precisa y adaptativa de la información relevante, mientras que el procesamiento de eventos discretos proporcionará un marco para el análisis de patrones y la toma de decisiones. Al desarrollar una solución que combine estos elementos, se espera mejorar la capacidad de las máquinas inteligentes para comprender y responder a la información compleja y dinámica.

1.3 Objetivos

1.3.1 Objetivo General

Desarrollar un modelo de memoria artificial que emule el funcionamiento intrincado de la memoria humana. Este objetivo conlleva la búsqueda de una estructura capaz de imitar las complejas habilidades cognitivas presentes en la retención, asociación y recuperación de información que son emblemáticas del cerebro humano.

1.3.2 Objetivos Específicos

- Explorar las características clave de los módulos de atención y la red sensorial en relación con la memoria artificial.
- Realizar una integración efectiva con los módulos de atención y la red sensorial.
- Identificar posibles mejoras para el modelo propuesto.
- Identificar las limitaciones del modelo propuesto.

1.4 Propósitos

- Explorar conceptos básicos de integración de una memoria artificial con módulos de atención y una red sensorial para mejorar la capacidad de adaptación de una máquina inteligente de manera sencilla.
- Contribuir al conocimiento y comprensión de cómo se pueden conectar componentes clave en máquinas inteligentes de bajo alcance, como una memoria artificial con módulos de atención y una red sensorial.

- Proporcionar una base de trabajo para futuras investigaciones y desarrollos en el ámbito de la integración de módulos de atención y redes sensoriales con memorias artificiales en máquinas inteligentes de mayor complejidad.
- Demostrar la aplicabilidad y utilidad de la integración propuesta en un contexto limitado, destacando su potencial para mejorar la capacidad de respuesta y adaptación de máquinas inteligentes básicas.

Capítulo 2

Marco Teórico

2.1 Sistema

Un sistema se puede definir como una colección de componentes o elementos interconectados que trabajan juntos para lograr un objetivo o propósito común (Gibson et al., 2005). Estos componentes pueden ser entidades físicas, como máquinas u organismos, o entidades abstractas, como información o estructuras sociales. Los sistemas pueden exhibir propiedades emergentes que surgen de las interacciones y relaciones entre sus componentes (Waguespack & Schiano, 2013). El comportamiento de un sistema está influenciado por su dinámica interna, su entorno externo y los bucles de retroalimentación que existen dentro del sistema. Comprender y analizar los sistemas es crucial en diversos campos, incluyendo ingeniería, biología, sociología y sistemas de información.

2.2 Memoria artificial

La memoria artificial se refiere a la capacidad de sistemas no biológicos para almacenar información y recuperarla según sea necesario. Los sistemas de memoria artificial pueden ser diseñados para imitar la capacidad del cerebro humano para almacenar y recordar recuerdos (Garner et al., 2012). También puede referirse a sistemas que utilizan mecanismos como células de memoria para recordar el pasado (Tan et al., 2002). El concepto de memoria artificial puede tener una amplia gama de aplicaciones, desde la inteligencia artificial hasta las memorias visuales artificiales orgánicas y los robots humanoides.

2.3 Recuperación de información

La recuperación de datos se refiere al proceso de acceder y extraer información relevante de un conjunto de datos o base de datos (Pao, 1993). Implica buscar o consultar datos específicos basados en criterios predefinidos y recuperar la información deseada para su análisis o uso posterior (Fernández et al., 2014). En la recuperación de datos, la precisión y confiabilidad de los datos obtenidos son factores importantes. Se pueden emplear varios métodos y herramientas, como la recuperación semántica y la recuperación pragmática, en los procesos de recuperación de datos (Pao, 1993). Además, los avances en el aprendizaje automático y enfoques de recuperación de información han facilitado una mejor adquisición de conocimiento a partir de los datos (Fernández et al., 2014). En general, la recuperación de datos desempeña un papel crucial en la obtención y utilización de datos para la investigación, la toma de decisiones y diversas aplicaciones en diferentes campos.

2.4 Maquina autónoma

Una máquina autónoma se puede definir como una máquina o sistema capaz de operar y tomar decisiones sin intervención o control humano (Werkhoven et al., 2018). Se refiere a máquinas que tienen la capacidad de realizar tareas, adaptarse a cambios en su entorno y tomar decisiones basadas en sus propios algoritmos o programación (Dittrich et al., 2019). Las máquinas autónomas pueden adquirir nuevas habilidades y comportamientos sin intervención humana, demostrando un comportamiento autónomo en su desarrollo (Isaka, 2023). Sin embargo, se debe tener en cuenta que las máquinas autónomas actuales pueden tener limitaciones para manejar fallas y situaciones inesperadas (Werkhoven et al., 2018). Estas máquinas aprovechan tecnologías como el aprendizaje automático, la inteligencia artificial y el Internet de las cosas para operar y tomar decisiones inteligentes sin control humano, mejorando la eficiencia y productividad en diferentes ámbitos.

2.5 Computación cognitiva

La computación cognitiva se refiere a sistemas que aprenden, razonan e interactúan con los humanos de manera natural (Furbach et al., 2019). Combina elementos de la inteligencia artificial y el aprendizaje automático para mejorar el rendimiento humano-computadora (Gerhardt-Powals, 1996). Los modelos de computación cognitiva tienen como objetivo comprender y simular los procesos cognitivos humanos, como la percepción, la atención, la memoria y la toma de decisiones (Wang, 2011). En el contexto de los entornos inteligentes y el Internet de las cosas, la computación cognitiva se puede aplicar a dispositivos cognitivos, redes y análisis cognitivos (Alsamhi et al., 2021). En general, la computación cognitiva desempeña un papel crucial en la comprensión y explicación de diversos fenómenos, desde la vida cotidiana hasta los dominios científicos (Thagard & Litt, 2001).

2.6 Simulación discreta

La simulación discreta se refiere a la modelización y análisis de sistemas utilizando técnicas de simulación de eventos discretos, donde los eventos ocurren en momentos distintos en el tiempo y tienen un impacto significativo en el comportamiento del sistema (Dietz et al., 2005). Implica representar el comportamiento y las interacciones de los componentes del sistema, así como el tiempo y la secuencia de los eventos, utilizando variables discretas y distribuciones de probabilidad (Casas et al., 2013). Al simular los eventos discretos y sus efectos, la simulación discreta permite el análisis y la optimización de sistemas complejos, proporcionando información sobre su rendimiento, comportamiento y procesos de toma de decisiones.

2.7 Programación orientada a objectos

La programación orientada a objetos (OOP, por sus siglas en inglés) es un paradigma de programación que organiza el diseño de software alrededor de objetos, que son instancias de clases que encapsulan datos y comportamiento (Collier & O'Hare, 2009; Iqbal & Allen, 2010). En la OOP, los objetos interactúan entre sí a través de métodos, que son funciones o procedimientos asociados con la

clase. Los principios clave de la OOP incluyen encapsulación, herencia y polimorfismo. La encapsulación permite agrupar datos y métodos dentro de un objeto, proporcionando ocultamiento de datos y abstracción. La herencia permite la creación de nuevas clases heredando propiedades y comportamientos de clases existentes, fomentando la reutilización de código y la modularidad. El polimorfismo permite tratar objetos de diferentes clases como instancias de una superclase común, lo que facilita la flexibilidad y la extensibilidad.

La OOP proporciona varios beneficios, incluyendo modularidad, reutilización, mantenibilidad y escalabilidad. Promueve una forma más intuitiva y natural de modelar entidades y relaciones del mundo real, lo que facilita la comprensión y el mantenimiento del código. La OOP se utiliza ampliamente en diversos lenguajes de programación, como Java, C++ y Python, y es especialmente adecuada para el desarrollo de software a gran escala y sistemas complejos.

2.8 No determinismo

El no determinismo se refiere a una propiedad de sistemas o procesos en los que la elección del siguiente paso o resultado no está determinada de manera única por el estado actual o las entradas (Hughes et al., 2016; Leemans & Polyvyanyy, 2020). Introduce múltiples resultados o comportamientos posibles, y el resultado real puede depender de factores como elecciones internas, eventos probabilísticos o conflictos (Kempf et al., 2013; Gutiérrez, 2015; Ward & Zedan, 2016). El no determinismo se encuentra comúnmente en diversos dominios, como sistemas distribuidos, concurrencia, pruebas y verificación (Fraser & Wotawa, 2007). Presenta desafíos en cuanto a comprender y predecir el comportamiento del sistema, así como diseñar algoritmos efectivos y estrategias de prueba. Comprender y gestionar el no determinismo es crucial para garantizar la confiabilidad, corrección y rendimiento de los sistemas en diversos dominios de aplicación.

2.9 Metodología Kanban

La metodología Kanban es un enfoque ágil de gestión de proyectos que enfatiza la visualización del trabajo, la limitación del trabajo en progreso y la optimización de la eficiencia del flujo de trabajo

(Matthies, 2018; Biloskurskyi, 2022; Cawley et al., 2013). Se originó a partir del sistema de programación Kanban desarrollado en Toyota y se ha adaptado a diversos ámbitos, incluyendo el desarrollo de software, la fabricación y la logística (Mourato et al., 2020). En Kanban, los elementos de trabajo se representan como tarjetas en un tablero Kanban, que proporciona una representación visual de las etapas del flujo de trabajo y el estado de cada tarea (Cawley et al., 2013). La metodología promueve un sistema basado en la demanda, donde el nuevo trabajo se incorpora al flujo de trabajo solo cuando hay capacidad, evitando sobrecargas y cuellos de botella. Kanban permite a los equipos tener una comprensión clara de su trabajo, identificar y abordar ineficiencias del proceso y mejorar continuamente su flujo de trabajo. Ofrece flexibilidad, adaptabilidad y transparencia, lo que lo hace adecuado tanto para proyectos pequeños como para proyectos a gran escala (Biloskurskyi, 2022; Mourato et al., 2020).

2.10 Neurociencia

La neurociencia es el estudio del origen, naturaleza y significado funcional de los patrones complejos de actividad neural en el cerebro. Comprende la investigación de varios aspectos de la dinámica cerebral, incluyendo las oscilaciones colectivas, la sincronización neural y las avalanchas neuronales. La neurociencia también implica comprender las propiedades estructurales y dinámicas de la corteza, que subyacen a funciones como la acción, la percepción, el aprendizaje, el lenguaje y la cognición (Deco et al., 2008). Además, la neurociencia explora las capacidades computacionales de las neuronas y su papel en el procesamiento de la información (Timme et al., 2016). El campo de la neurociencia utiliza diversas técnicas, incluyendo métodos electrofisiológicos y de neuroimagen, para estudiar la actividad cerebral y su relación con el comportamiento y la cognición (Anglada-Tort & Skov, 2022; Grootswagers et al., 2017).

2.11 Memoria

La memoria se refiere al proceso cognitivo de codificar, almacenar y recuperar información a lo largo del tiempo (Woodman et al., 2001). Implica la formación y el mantenimiento de representaciones de experiencias pasadas, conocimientos y habilidades. Los procesos de memoria son complejos e

involucran diversos mecanismos cognitivos y neurales. La eficiencia de los procesos de memoria puede verse influenciada por factores como la atención, la carga cognitiva y la excitación emocional (Thomsen & Hansen, 2009).

2.12 Memoria Asociativa

La memoria asociativa se refiere a la capacidad del cerebro de vincular o asociar diferentes fragmentos de información o conceptos entre sí. Implica la formación de conexiones o asociaciones entre elementos o ideas relacionadas en la memoria. Estas asociaciones pueden basarse en la similitud, el contexto o las experiencias previas. La memoria asociativa desempeña un papel crucial en diversos procesos cognitivos, como la creatividad, la memoria semántica, la memoria prospectiva y la memoria episódica. Por ejemplo, en la creatividad, las personas con una organización más flexible de los conceptos en su memoria semántica pueden recuperar asociaciones remotas más fácilmente (Ovando-Téllez et al., 2021). En la memoria prospectiva, las interrupciones y los cambios de tarea pueden interferir en la capacidad para recordar y ejecutar acciones previstas (Finstad et al., 2006). En la memoria episódica, la asociación de objetos, ubicación e información contextual es necesaria para formar una memoria integrada de un evento. Comprender los mecanismos y funciones de la memoria asociativa puede proporcionar ideas sobre cómo se almacena, recupera y utiliza la información en el cerebro.

2.13 Memoria semántica

La memoria semántica se refiere al sistema de memoria a largo plazo que almacena conocimientos generales y conceptos sobre el mundo, incluyendo hechos, significados y relaciones entre diferentes conceptos. Es responsable de nuestra comprensión del lenguaje, los objetos y el mundo que nos rodea. La memoria semántica desempeña un papel crucial en diversos procesos cognitivos, como la comprensión del lenguaje, la categorización y la resolución de problemas. Por ejemplo, la memoria semántica nos permite entender el significado de las palabras y las frases (Gordon et al., 2001), recuperar información de la memoria durante tareas de fluidez verbal y organizar y recordar información relevante para tareas específicas (Kersten & Murphy, 2015).

2.14 Memoria prospectiva

La memoria prospectiva se refiere a la capacidad de recordar y llevar a cabo acciones previstas en el futuro (Kliegel et al., 2008). Implica recordar realizar una tarea o acción específica en un momento determinado o en respuesta a una señal específica. La memoria prospectiva desempeña un papel crucial en diversos aspectos de la vida diaria, como recordar asistir a citas, tomar medicamentos o completar tareas en momentos específicos.

La investigación sobre la memoria prospectiva ha crecido en los últimos años debido a su importancia teórica e implicaciones prácticas en campos como la psicología cognitiva, el envejecimiento, la psicología clínica y la neurología (Kliegel et al., 2008). Numerosos estudios han investigado diferentes aspectos de la memoria prospectiva, incluyendo sus procesos cognitivos, mecanismos neuronales, cambios en el desarrollo y el impacto de diversos factores como el envejecimiento, el estrés y los trastornos neurológicos.

2.15 Memoria episódica

La memoria episódica se refiere al sistema de memoria a largo plazo responsable de almacenar y recuperar experiencias personales y eventos específicos que están vinculados a un momento y lugar particular (Tulving, 2002). Permite a las personas recordar eventos pasados, como experiencias personales, detalles autobiográficos e información contextual asociada con esos eventos.

La memoria episódica implica la evocación de detalles específicos y la capacidad de viajar mentalmente en el tiempo hacia eventos pasados. Permite a las personas recordar no solo lo que sucedió, sino también dónde y cuándo ocurrió.

2.16 Memoria a corto plazo

La memoria a corto plazo se refiere a la capacidad de un individuo para almacenar y mantener información durante un breve período de tiempo. Es distinta de la memoria de trabajo, que implica tanto

almacenar como manipular información. La memoria a corto plazo está involucrada en diversos procesos cognitivos, como el cambio de tareas (Schneider & Logan, 2005), la generación de ideas (Luo & Toubia, 2015), la memoria del orden (Clarkson et al., 2016), la retención del lenguaje (Köpke & Nespoulous, 2006) y la memoria visual (Ang & Lee, 2010). Se considera un predictor válido del desempeño laboral y de capacitación. La memoria a corto plazo forma parte de la arquitectura cognitiva, que también incluye la memoria sensorial y la memoria a largo plazo. Aunque existen debates sobre la capacidad de la memoria a corto plazo, se acepta generalmente que está limitada tanto en tiempo como en capacidad.

2.17 Memoria a largo plazo

La memoria a largo plazo se refiere a la capacidad de un individuo para almacenar y recuperar información a lo largo de un período extendido de tiempo. Implica la consolidación de memorias a través de procesos como la consolidación sináptica, la consolidación del sistema y la reconsolidación Específicamente, la consolidación sináptica se facilita mediante la potenciación a largo plazo, lo que conduce a la formación de conexiones sinápticas fuertes y duraderas (Chang et al., 2011). La consolidación del sistema ocurre durante días o semanas e implica la reorganización de los circuitos cerebrales para estabilizar las memorias dentro del sistema.

Investigaciones han demostrado que existen similitudes y diferencias entre la memoria de trabajo y la memoria a largo plazo (Rose et al., 2010). Mientras que la profundidad del procesamiento tiene efectos mínimos en la memoria de trabajo, tiene un impacto significativo en la memoria posterior en la memoria a largo plazo, especialmente el procesamiento semántico (Rose et al., 2010). Sin embargo, hay evidencia de que las tareas de memoria de trabajo implican la recuperación de la memoria secundaria, lo que sugiere cierta superposición entre los procesos de memoria de trabajo y memoria a largo plazo (Rose et al., 2010).

2.18 Memoria de trabajo

La memoria de trabajo se refiere al sistema cognitivo responsable de retener y manipular temporalmente la información durante tareas cognitivas complejas (Hitch et al., 2001). Implica el mantenimiento activo y la manipulación de información frente a distracciones o interferencias (Farrell et al., 2016). La capacidad de la memoria de trabajo es limitada y puede variar entre individuos (Lépine et al., 2005). Juega un papel crucial en diversos procesos cognitivos, como la atención, resolución de problemas, toma de decisiones y comprensión del lenguaje.

El concepto de memoria de trabajo ha sido estudiado ampliamente en la psicología cognitiva. A menudo se evalúa mediante tareas de complejidad secuencial que involucran tanto el almacenamiento como el procesamiento de información (Lépine et al., 2005). La memoria de trabajo se distingue de la memoria a corto plazo, ya que implica no solo el almacenamiento temporal de información, sino también la manipulación activa y actualización de esa información.

Capítulo 3
Marco Metodológico

3.1 Tipo de Investigación

En este proyecto, se ha optado por utilizar una metodología de investigación aplicada, siguiendo la clasificación propuesta por Kothari (2004), que distingue diversos tipos de investigación, como investigación básica, aplicada, exploratoria y descriptiva, entre otros. El enfoque seleccionado, la investigación aplicada, tiene como propósito principal aplicar los conocimientos científicos y teóricos obtenidos a través de la investigación básica para resolver problemas prácticos del mundo real.

El objetivo central es el diseño y desarrollo de una memoria artificial denominada "Historia y Episodio de Vida", capaz de emular las funciones cognitivas de la memoria humana, aprovechando la investigación aplicada como una herramienta para generar resultados concretos y aplicables en diferentes campos de la inteligencia artificial.

3.2 Diseño de la Investigación

El diseño de la investigación se basa en una metodología explicativa con el objetivo primordial de indagar en los aspectos esenciales relacionados con la memoria. En lugar de enfocarse en demostrar conclusiones absolutas, el enfoque se dirige hacia una comprensión más profunda y completa del problema en su contexto más amplio.

Para lograr este propósito, se ha llevado a cabo un análisis minucioso de las diferentes dimensiones que conforman la memoria en el cerebro humano. La exploración de los fragmentos de información más pequeños resulta crucial para cumplir con los objetivos planteados en este estudio.

La investigación explicativa ha sido desarrollada mediante grupos de discusión, lo que ha permitido obtener una comprensión enriquecida del tema de estudio.

La importancia de este enfoque radica en la posibilidad de adquirir una comprensión profunda de las complejidades y conexiones subyacentes de la memoria, lo que aporta un valor significativo al abordaje de futuros temas y desafíos en el contexto de la memoria artificial.

3.3 Metodología

Para lograr el propósito planteado, se ha empleado la metodología Kanban, la cual proporciona una estructura ágil y eficiente para la gestión de proyectos. Kanban se caracteriza por su enfoque en la optimización del flujo de trabajo, la planificación iterativa y la entrega continua de funcionalidades, adaptándose a las necesidades del proyecto.

Esta metodología se basa en el principio de visualizar y controlar el flujo de trabajo a través de un tablero visual, dividido en columnas que representan diferentes etapas del proceso. En este caso, se han definido las siguientes columnas: "Pendiente", "En progreso" y "Completado". Cada una de ellas contiene tarjetas que representan las tareas a realizar, las cuales son movidas de una columna a otra a medida que avanzan en su ejecución.

3.3.1 Planificación

La etapa inicial del proyecto se enfocó en un análisis exhaustivo de los requisitos y especificaciones del proyecto, identificando las funcionalidades clave que debían implementarse para alcanzar los objetivos

establecidos. Con la información recopilada, se crearon tarjetas en la columna "Pendiente" del tablero Kanban, representando las tareas a realizar en cada iteración. Este enfoque iterativo facilitó la obtención de resultados concretos en cada ciclo del proceso y permitió establecer criterios de éxito para evaluar el rendimiento del sistema.

Las iteraciones se planificaron de manera detallada, definiendo un conjunto específico de tareas considerando la complejidad de las funcionalidades, los recursos disponibles y plazos. Esta planificación incremental permitió obtener avances concretos en cada etapa del proceso.

3.3.2 Ejecución

El desarrollo del proyecto se llevó a cabo en iteraciones, cada una enfocada en un conjunto específico de tareas. Kanban proporcionó un marco visual y flexible para controlar el flujo de trabajo, permitiendo una gestión constante del avance mediante actualizaciones en el tablero. Esta aproximación adaptativa fue esencial para asegurar un progreso eficiente del proyecto, manteniendo una entrega continua y permitiendo revisiones periódicas del progreso.

El enfoque de seguimiento y control permitió mantener un ritmo óptimo de trabajo y asegurar que el proyecto avanzara de acuerdo con los plazos y objetivos establecidos. Las actualizaciones periódicas en el tablero garantizaron claridad y visibilidad del progreso de cada tarea, lo que facilitó la comunicación interna.

3.3.3 Evaluación

Una vez que todas las tarjetas de una iteración se movieron a la columna "Completado", se realizó una revisión exhaustiva de los resultados obtenidos, incluyendo pruebas funcionales, verificación de

requisitos y análisis de desempeño. Las lecciones aprendidas y los resultados de estas revisiones permitieron realizar mejoras y ajustes necesarios antes de avanzar a la siguiente iteración.

En resumen, la metodología Kanban fue clave en la planificación, ejecución y evaluación del proyecto. Su enfoque ágil y adaptable permitió una gestión eficiente del flujo de trabajo, asegurando la entrega continua de funcionalidades y una revisión constante de los resultados obtenidos en cada etapa, garantizando el cumplimiento de los objetivos planteados para este proyecto.

Capítulo 4
Desarrollo del Proyecto

En este capítulo, se presentarán y desarrollarán las ideas relacionadas con la memoria artificial. Se destacarán los fundamentos del proyecto, enfocándose en los requisitos y especificaciones clave que dirigirán todo el proceso.

4.1 Planificación

El enfoque adaptado para la planificación de este proyecto se centra en el Debate Socrático, una aproximación que ha resultado fundamental para definir y detallar las funcionalidades clave en los proyectos que conforman la máquina inteligente. En este proceso, los integrantes del equipo del BrainCEMISID y nuestro Tutor participamos en discusiones profundas y reflexivas.

A través del Debate Socrático, se consiguió desglosar cada funcionalidad en sus componentes esenciales, explorar las implicaciones y llegar a definiciones precisas. La interacción entre los tesistas ha enriquecido el proceso al incorporar diversas perspectivas y enfoques. Además, la orientación experta de nuestro Tutor ha sido crucial para mantenernos enfocados en nuestros objetivos.

Este enfoque no solo ha impulsado los proyectos, sino que también nos ha equipado con habilidades críticas para el futuro. La capacidad de formular preguntas perspicaces y abordar problemas complejos será invaluable en nuestras trayectorias profesionales. En resumen, el Debate Socrático se ha convertido en una herramienta efectiva para nuestro trabajo en los proyectos que conforman la máquina inteligente

4.2 Integración de la Memoria Artificial, Módulo de Atención y Red Sensorial

El proyecto se enfoca en la implementación de un sistema autónomo que integra la memoria artificial, el módulo de atención y la red sensorial. Estos tres componentes representan proyectos individuales que convergen para formar un sistema coherente y funcional, capaz de mejorar la eficiencia y efectividad del proceso de almacenamiento y recuperación de información.

El objetivo principal es diseñar y desarrollar un sistema que emule las funciones cognitivas de la memoria humana. Se ha trabajado en la implementación de algoritmos y estructuras de datos que permitan el almacenamiento y recuperación eficiente de la información, con el fin de lograr un funcionamiento similar al de la memoria humana.

Por otro lado, el proyecto del módulo de atención (Albarrán, 2023) se ha enfocado en el diseño y desarrollo de algoritmos y técnicas que permitan seleccionar y enfocarse en los elementos relevantes dentro de la memoria. Este módulo desempeña un papel crucial al dirigir la atención hacia la información más relevante, mejorando así la eficiencia y la capacidad de procesamiento del sistema en general.

Simultáneamente, el módulo de la red sensorial (Eyzell, 2023) se ha dedicado a capturar y procesar las entradas provenientes del entorno. Se han empleado técnicas y algoritmos especializados para interpretar y extraer información relevante de estos datos. La incorporación de este módulo en el sistema habilita a la memoria para interactuar con el entorno y adaptarse a los estímulos externos, enriqueciendo así la base de conocimientos y la capacidad de respuesta del sistema.

En conjunto, estos tres proyectos convergen para formar una máquina autónoma y funcional. La integración de la memoria artificial, el módulo de atención y la red sensorial permite una interacción fluida y coherente entre los componentes. A través de la colaboración y el trabajo conjunto, se busca lograr un sistema autónomo capaz de aprender, almacenar y procesar información de manera eficiente y adaptativa, emulando así las capacidades cognitivas de la memoria humana.

4.3 Diseño e Implementación de la Arquitectura de la Memoria Artificial

En la concepción de la arquitectura de la memoria artificial, se ha empleado una tabla de dispersión en Python para gestionar los recuerdos. Cada recuerdo se representa como con un descriptor que actúa como identificador único y permite un acceso eficiente a los recuerdos específicos.

Además del descriptor, cada recuerdo contiene una lista de recuerdos asociados, lo que facilita las conexiones y relaciones entre los diferentes elementos de la memoria. Estas asociaciones entre recuerdos permiten una recuperación y procesamiento más eficiente de información interconectada, imitando la forma en que los humanos asocian y recuperan recuerdos en su memoria.

La arquitectura también incorpora los sentidos proporcionados por la red sensorial, emulando así los sentidos humanos en el proceso de almacenamiento y recuperación de información. Cada recuerdo en la memoria se asocia a un sentido específico, lo que permite establecer relaciones entre la información capturada por la red sensorial y los recuerdos almacenados. Esta integración busca enriquecer la experiencia de la memoria y su capacidad para interpretar y procesar datos sensoriales del entorno.

stats

Para acceder a los detalles completos y a la implementación integral de este proyecto, se invita a visitar el siguiente enlace al repositorio correspondiente: https://github.com/MarAlba8/BRAIN_CEMISID_4.0. El repositorio proporciona una visión detallada del código y los recursos relacionados con la arquitectura de la memoria artificial y su integración con la red sensorial y módulo de atención.

En las siguientes subsecciones, se describirán los elementos clave que conforman la arquitectura de este proyecto. Desde los componentes esenciales de la memoria artificial hasta la incorporación de los sentidos simulados, se presentarán los aspectos fundamentales que sustentan esta propuesta.

4.4 Historia de Vida

En el contexto de la memoria artificial, se reconoce que existen tres tipos fundamentales de memorias: la memoria emocional, la memoria biológica y la memoria cultural. La memoria emocional abarca las experiencias y recuerdos asociados a las emociones y estados afectivos. Registra los eventos y estímulos que generan respuestas emocionales significativas. La memoria biológica, por otro lado, almacena información relacionada con el funcionamiento biológico y los procesos fisiológicos. Contiene datos sobre el cuerpo, la salud y los aspectos biológicos de la experiencia humana. La memoria cultural se refiere a los conocimientos, creencias, valores y tradiciones compartidas por una comunidad o sociedad.

El conjunto de estas tres memorias, la emocional, biológica y cultural, se fusiona en lo que se denomina la "Historia de Vida". Es un reflejo completo y diverso de las experiencias y la identidad de un individuo. La historia de vida captura los momentos emocionales significativos, los aspectos biológicos relevantes y la influencia de la cultura en el desarrollo de una persona.

Los cambios de atención juegan un papel crucial. El sistema de memoria monitorea constantemente los cambios en la atención para determinar cuándo y en qué aspecto actualizar la historia de vida. Estos cambios de atención, impulsados por el módulo de atención, señalan qué aspecto de la historia de vida requiere una actualización: ya sea un evento emocional relevante, un cambio en el estado biológico o una nueva adquisición cultural.

De esta manera, la memoria se adapta y actualiza la historia de vida de acuerdo con los cambios en la atención, asegurando una representación precisa y completa de las experiencias. Al integrar los aspectos emocionales, biológicos y culturales en la historia de vida, la memoria artificial busca emular y comprender mejor el funcionamiento complejo de la memoria humana.

4.5 Episodio de Vida

Para complementar la arquitectura de la memoria artificial, se ha incorporado un buffer temporal denominado 'Episodio de Vida', que funciona como una memoria a corto plazo. Este buffer desempeña un papel crucial en la gestión de la información recibida de la red sensorial y su posterior almacenamiento en la memoria principal.

Cada vez que se recibe un nuevo evento o estímulo de la red sensorial, se registra en el Episodio de Vida correspondiente al sentido específico al que pertenece, capturando así una instantánea de la experiencia sensorial en un momento dado.

La existencia de un Episodio de Vida por cada sentido permite mantener una organización clara de los eventos sensoriales asociados a cada uno de ellos. Por ejemplo, los eventos auditivos se registran en el Episodio de Vida auditivo, garantizando una separación y clasificación adecuada de la información sensorial dentro de la memoria.

Cuando se produce un cambio en la atención del sistema, el contenido relevante del Episodio de Vida se traslada y almacena en la memoria principal de la memoria. Este proceso de transferencia se realiza para conservar y preservar la información significativa experimentada por la red sensorial, asegurando que los eventos relevantes se almacenen de manera permanente en la memoria.

La transferencia de los eventos del Episodio de Vida a la memoria principal sigue criterios de relevancia y atención. Los eventos seleccionados y enfocados por el módulo de atención son considerados de mayor importancia y se priorizan para su almacenamiento. Este enfoque está alineado con el objetivo de la memoria artificial de retener y procesar la información más relevante y significativa para la toma de decisiones y la generación de respuestas adecuadas.

Una característica importante del Episodio de Vida es que, si la atención no apunta hacia un evento almacenado en dicho buffer temporal, este se vaciará progresivamente. Es decir, los eventos registrados en el 'Episodio de Vida' que no sean seleccionados o enfocados por el módulo de atención para

su procesamiento y traslado a la memoria principal, se eliminarán gradualmente para dar espacio a nuevos eventos sensoriales.

4.6 Consultas a la memoria

La capacidad del módulo de atención para realizar consultas a las diferentes memorias constituye uno de los aspectos más destacados en esta arquitectura. Este módulo actúa como el puente entre la red sensorial y las memorias asociadas, permitiendo acceder y recuperar información de manera selectiva y eficiente. Al recibir una secuencia de recuerdos basados en el sentido consultado, el módulo de atención tiene la habilidad de enfocar su atención en aquellos eventos relevantes y significativos para el momento presente.

Cada sentido definido en el proyecto, incluyendo oído, tacto, vista, olfato, gusto, cuerpo y tiempo, posee su propio conjunto de recuerdos asociados. El módulo de atención puede explorar y navegar por estas distintas memorias, en busca de información específica relacionada con cada sentido. Por ejemplo, al consultar la memoria auditiva, el módulo de atención accederá a los eventos y experiencias sonoras previamente almacenados, lo que permitirá identificar patrones acústicos, reconocer sonidos familiares y recordar eventos pasados vinculados a esta modalidad sensorial.

La posibilidad de realizar consultas en cualquiera de los sentidos proporciona a la memoria una versatilidad notable en la interpretación y comprensión del entorno. Cuando el módulo de atención se enfoca en un sentido determinado, se enriquece la experiencia de la memoria al permitirle emular el enfoque sensorial humano. Esto resulta en un sistema más adaptable y sensible a los estímulos externos, lo que, a su vez, enriquece la base de conocimientos y la capacidad de respuesta del sistema.

La representación gráfica proporcionada en la figura 1.1 permite observar la estructura subyacente de la máquina inteligente, donde la sección delineada constituye el módulo de la memoria artificial. Esta delimitación es crucial para comprender cómo esta máquina incorpora y organiza la función esencial de la memoria en su funcionamiento global.

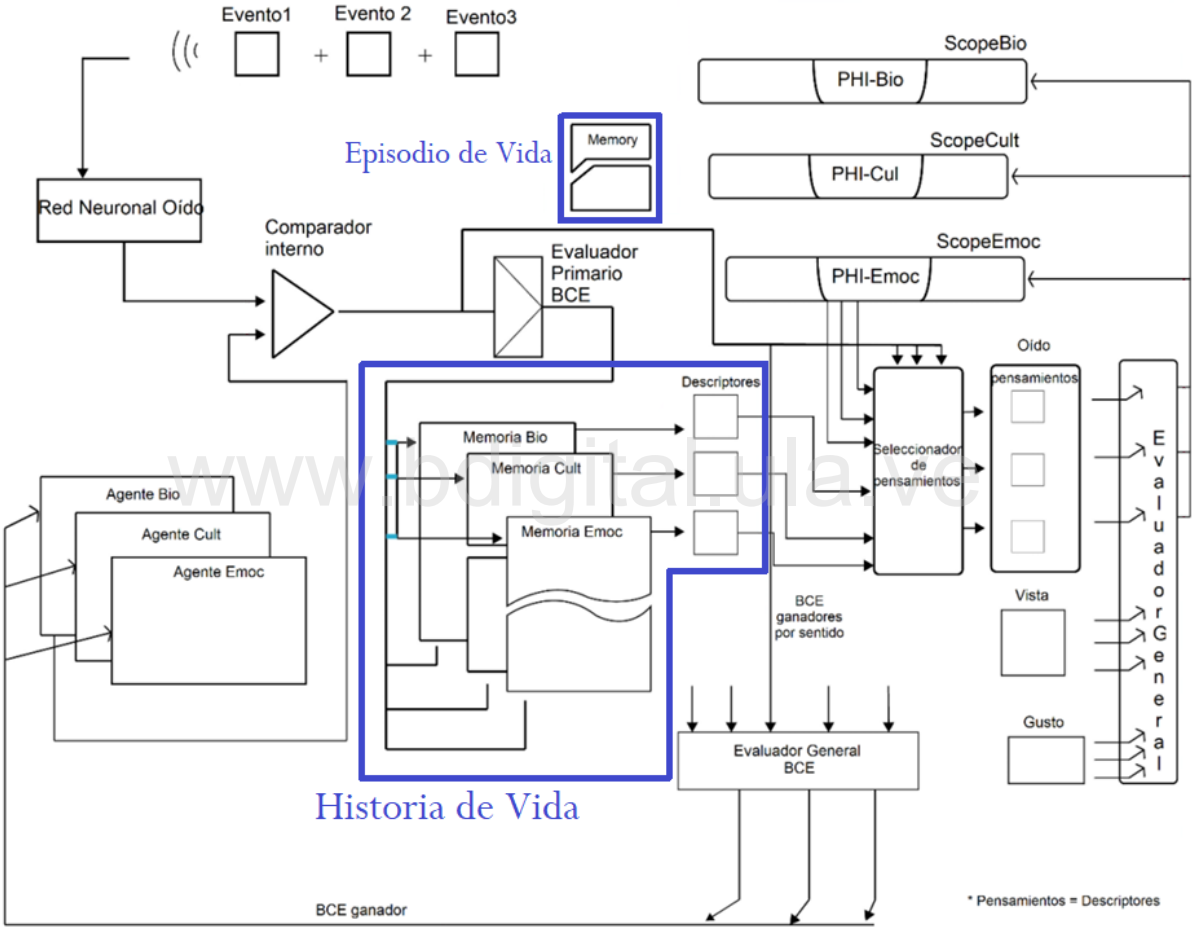

Figura 1.1 Modelo de la máquina inteligente

4.7 Hitos y Evaluación en el Desarrollo de la Memoria Artificial

Durante el desarrollo de este proyecto se establecieron hitos importantes que representaban puntos clave en el avance del proyecto. Estos hitos se utilizaron como referencia para evaluar el progreso y medir el logro de objetivos específicos en diferentes etapas del desarrollo. A continuación, se describen algunos ejemplos de hitos relevantes en el proceso de desarrollo de la memoria artificial:

1. Implementación básica de la estructura de memoria: Este hito marcó la culminación de la implementación inicial de la estructura de la memoria. Se estableció una base funcional que permitía el almacenamiento y recuperación de información en la memoria, utilizando la tabla de dispersión y los descriptores de recuerdos como elementos fundamentales.

2. Integración del módulo de atención: En este hito, se logró la integración exitosa del módulo de atención. Esto incluyó la implementación de algoritmos y técnicas que permitían la selección y enfoque en elementos relevantes dentro de la memoria, mejorando así la capacidad de procesamiento y toma de decisiones del sistema.

3. Incorporación de la red sensorial: En este hito, se logró la integración exitosa de la red sensorial en la arquitectura de la memoria. Se implementaron algoritmos y técnicas especializados para capturar y procesar los datos sensoriales provenientes del entorno, enriqueciendo así la base de conocimientos y la capacidad de respuesta del sistema.

4. Validación de la funcionalidad de almacenamiento y recuperación: En este hito, se realizó una validación exhaustiva de la funcionalidad de almacenamiento y recuperación de la memoria. Se llevaron a cabo pruebas funcionales para verificar la precisión y eficiencia del sistema en la gestión de la información almacenada y su posterior recuperación.

5. Evaluación del rendimiento del sistema: Este hito implicó la evaluación y medición del rendimiento del sistema de memoria artificial. Se realizaron pruebas y análisis de desempeño para evaluar la eficiencia, velocidad y capacidad de respuesta del sistema en diferentes escenarios y condiciones.

4.8 Simulación

Con el propósito de poder representar el flujo completo de los módulos antes mencionados, se realizó una interfaz gráfica donde se pueda apreciar de una mejor manera los resultados que se originan por la interacción de los módulos. Esta interfaz proporciona una visualización clara y detallada de cada etapa del proceso, permitiendo una comprensión más profunda de cómo se interconectan los distintos componentes y cómo influyen en los resultados finales. Además, la interfaz ofrece herramientas interactivas que facilitan la exploración y análisis de los datos generados, brindando una experiencia intuitiva y enriquecedora para el usuario.

La representación visual presentada en la figura 1.2 brinda una visión detallada de los componentes esenciales que constituyen la Historia de Vida. En la parte izquierda de la imagen, se destacan las tres memorias fundamentales que dan forma a esta historia: la memoria biológica, la memoria cultural y la memoria emocional.

En la zona derecha de la ilustración, se muestra el Episodio de Vida, y debajo de este, una serie de opciones interactivas dispuestas dentro de la interfaz. Entre estas opciones se encuentran: "Add Memory", que posibilita la inclusión de nuevos elementos en alguna o todas las memorias de la Historia de Vida; "Fill Life Episode", que permite cargar una descripción detallada del evento, que se está procesando actualmente por la red sensorial, a la memoria de corto plazo; "Query Memory", que da acceso a los recuerdos almacenados; y "Handle Attention", que sugiere una intervención en la dirección de la focalización mental.

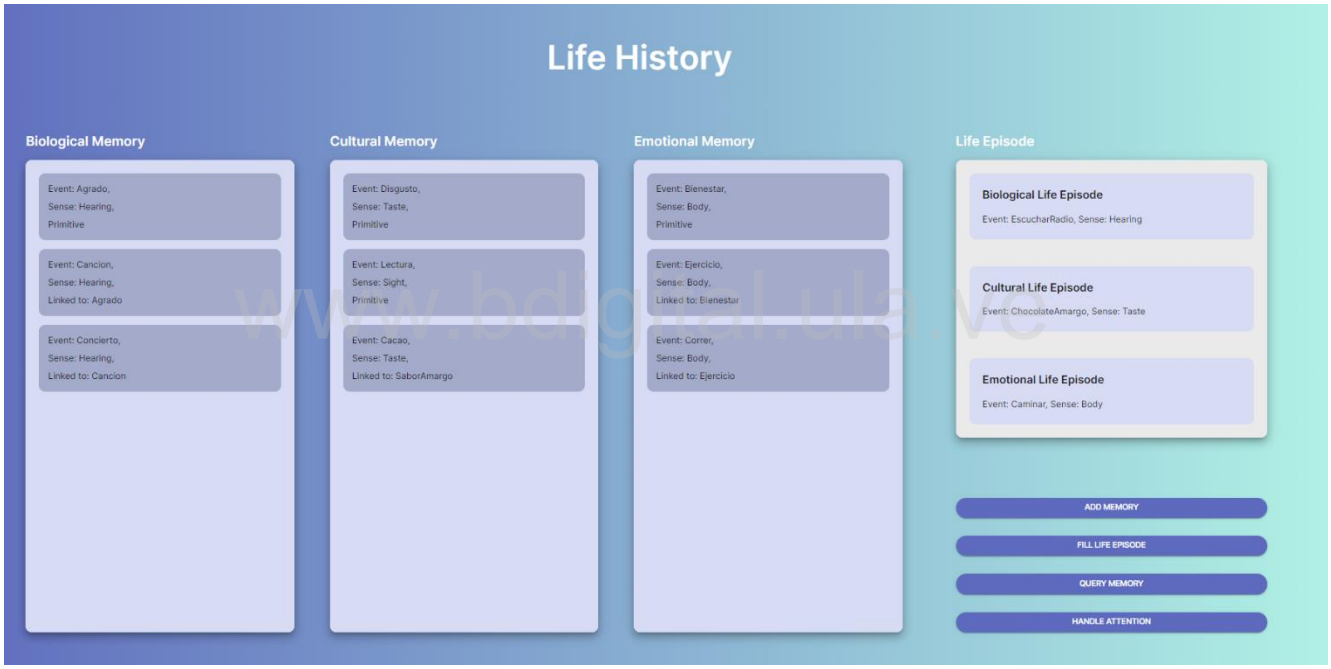

Figura 1.2 Interfaz gráfica de la simulación

Inicialmente, se cargan a la memoria algunas experiencias y pensamientos primitivos que serán de utilidad para darle significado a nuevos recuerdos de la memoria. Estos pensamientos primitivos están generados por el módulo de la red sensorial, el cual actúa como una puerta de entrada al sistema, capturando información sensorial del entorno y transformándola en representaciones iniciales. Estas representaciones, junto con las experiencias almacenadas previamente, sirven como cimientos sobre los cuales se construirán y relacionarán los nuevos recuerdos y aprendizajes. El módulo de la red sensorial desempeña un papel fundamental en el proceso de adquisición y organización de la información, permitiendo al sistema establecer conexiones y asociaciones significativas entre los datos sensoriales entrantes y el conocimiento existente en la memoria. De esta manera, se logra una comprensión más profunda y contextualizada de las experiencias, lo que contribuye a enriquecer la capacidad de la memoria para capturar y retener la diversidad y complejidad del entorno.

La figura 1.3 ofrece una representación visual de la pantalla diseñada para agregar un recuerdo a la Historia de Vida. Dentro de esta interfaz, se presentan diversos campos relevantes que capturan los detalles esenciales del recuerdo que se está a punto de incorporar. Estos campos incluyen, en primer lugar, un espacio destinado al descriptor del evento, permitiendo una breve descripción del suceso en cuestión. Además, se proporciona un campo dedicado a identificar el sentido sensorial al que el evento está vinculado.

Uno de los aspectos notables es la opción de asignar a qué tipo de memoria se asociará el recuerdo recién agregado. Esta función ofrece la flexibilidad de vincular el recuerdo a todas las memorias, lo que refleja una conexión integral entre las memorias biológica, cultural y emocional, o seleccionar una memoria específica en la que se depositará el recuerdo.

Otro campo relevante es aquel que permite indicar el número de neurona dentro de la red sensorial que procesó el evento en cuestión.

Por último, la interfaz exhibe una lista de eventos previamente registrados que están conectados con el recuerdo que se pretende conservar. Esta función de asociación permite enriquecer la contextualización de la memoria, trazando vínculos y relaciones entre diversos momentos de la historia de vida.

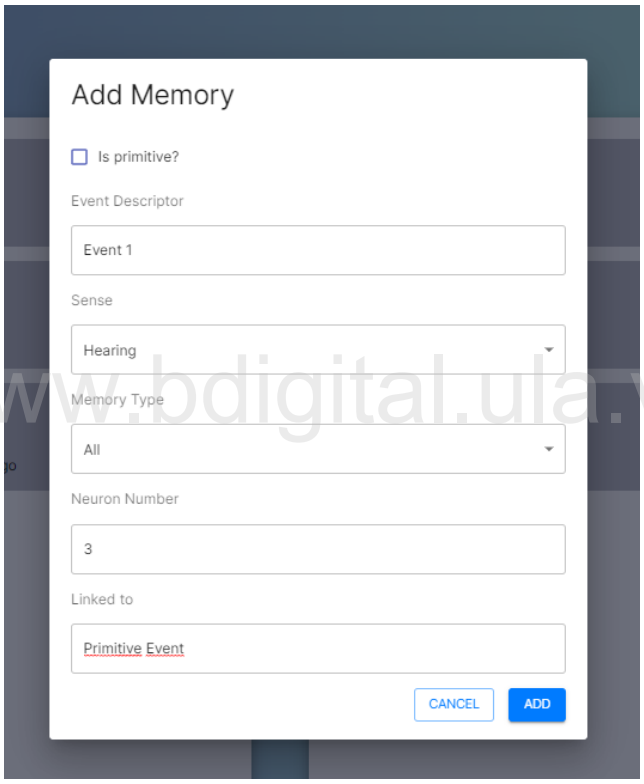

Figura 1.3 Cargar recuerdos a la Historia de Vida

Una vez que la memoria cuenta con elementos almacenados y ha iniciado el proceso, se vuelve accesible para su consulta cuando el módulo de atención lo requiere. Este módulo juega un papel crucial en la selección y enfoque hacia los recuerdos y experiencias relevantes en un momento determinado. Una vez activado, el módulo de atención puede acceder a cualquiera de las tres memorias según los diferentes sentidos: visual, auditiva, táctil, olfato, gusto, cuerpo y tiempo. Esta habilidad de consulta multisensorial permite al sistema tener un acceso completo y equilibrado a los recuerdos y experiencias almacenados en diversas modalidades sensoriales, potenciando así su capacidad para comprender el entorno y tomar decisiones adecuadas en diversas situaciones.

En la Figura 1.4, se ilustran las consultas clasificadas según el sentido y el tipo de memoria, ofreciendo una representación visual de cómo se organizan las búsquedas en función de estas dimensiones clave.

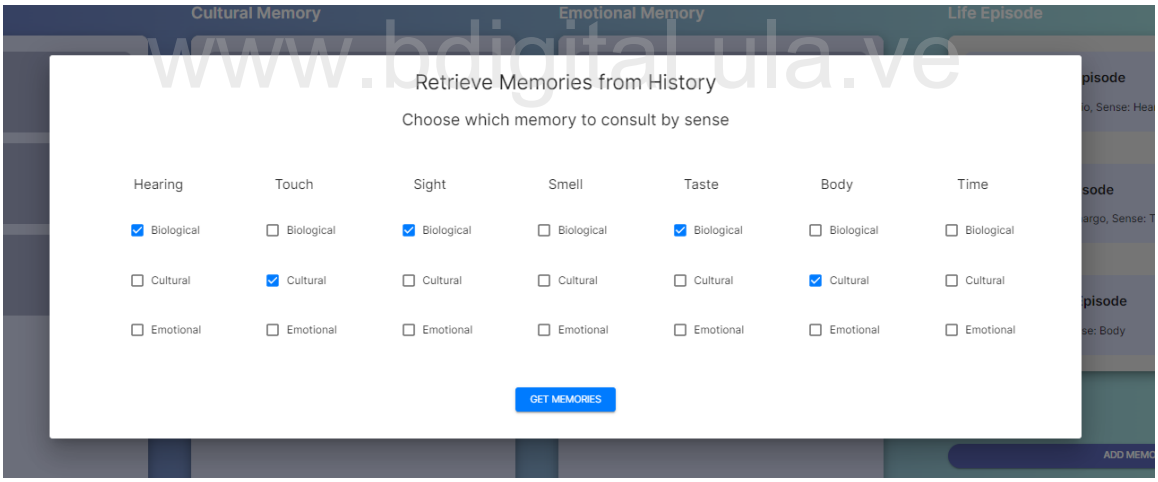

Figura 1.4 Consultas según el sentido y el tipo de memoria

Para determinar si un recuerdo debe ser preservado en la memoria, el módulo de atención lleva a cabo un proceso de evaluación mediante el envío de una secuencia de eventos. Durante este proceso, el módulo de atención juega un papel fundamental al dictar cuál es el factor a consultar en el episodio de vida o memoria de corto plazo. Si el módulo de atención muestra una mayor concentración en el evento específico destacado por este factor, dicho recuerdo se almacenará en la memoria. Esta interacción entre el módulo de atención y el episodio de vida garantiza que los recuerdos relevantes y significativos sean retenidos, asegurando así una memoria más eficiente y adaptativa para el funcionamiento del sistema cognitivo.

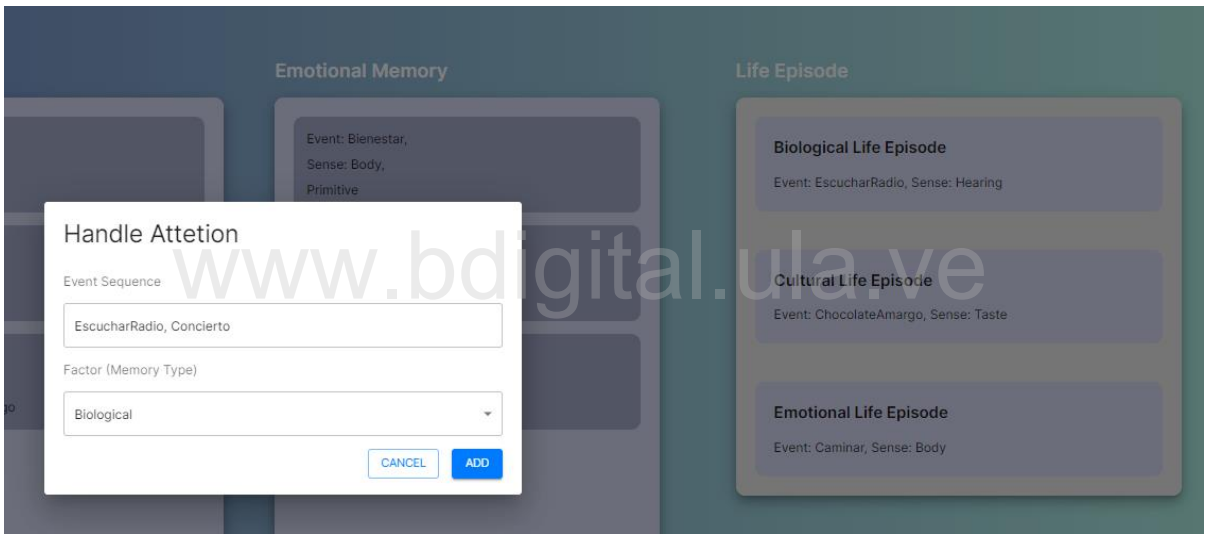

Figura 1.5 Manejo de atención.

A la derecha se encuentran los episodios de vida de cada factor (biológico, cultural y emocional)

A la izquierda se ubica la interfaz para recibir la secuencia de eventos por parte de la atención y escoger el factor a evaluar

4.9 Recorrido Detallado del Código

En esta subsección, se realizará un análisis en profundidad del código proporcionado. El código comprende dos clases interconectadas, Memory y History, que trabajan juntas para simular una memoria artificial y gestionar la historia de eventos. Cuando se mencionan patrones en el contexto de esta arquitectura, se hace referencia a pensamientos primitivos y emociones abstractas. Estos patrones actúan como elementos fundamentales en la memoria artificial, desempeñando un papel crucial en su inicialización. Los patrones sirven como cimientos sobre los cuales se construye la historia de eventos y recuerdos, otorgándoles significado y contexto.

La memoria artificial utiliza estos patrones como puntos de referencia para interpretar y contextualizar nuevos eventos. Al establecer conexiones entre los eventos actuales y los patrones iniciales, la memoria puede inferir significado y relación. Esto refleja el proceso humano de asociar experiencias previas con situaciones actuales, lo que resulta en una red interconectada de recuerdos.

4.9.1 Clase Memory

La clase Memory representa una memoria artificial capaz de almacenar eventos y su información asociada. En la siguiente sección, se detallan los atributos y métodos esenciales que conforman esta arquitectura, permitiendo una comprensión más profunda de su funcionamiento.

Los atributos desempeñan un papel fundamental en la configuración y operación de la memoria artificial. A continuación, se describen los atributos clave:

- `factor`: El atributo "factor" juega un papel fundamental durante la inicialización de la clase, siendo esencial para distinguir entre los diversos tipos de memoria, tales como biológica, emocional y cultural. Al crear una instancia de la clase Memory con un valor específico para este atributo, se establece una vinculación directa entre el tipo de memoria y la instancia en cuestión.

Esto facilita la gestión personalizada de distintos tipos de eventos y recuerdos de manera independiente.

- `life_history`: Un diccionario para almacenar eventos pasados y su información. Cada evento es representado por un diccionario que contiene su evento, sentido, número de neurona, lista de patrones asociados y eventos vinculados.

- `life_episode`: Un diccionario para episodios de vida. Cada episodio de vida es representado por un diccionario que contiene información similar al evento, incluyendo evento, sentido, número de neurona y lista de patrones.

- `stats`: Un diccionario para almacenar estadísticas de los eventos, organizadas por sentido y número de registros. Este atributo es de vital utilidad para el módulo de memoria, ya que permite llevar un registro detallado de los eventos y sus ocurrencias. Facilita el análisis y seguimiento de la importancia y frecuencia de los eventos en diferentes sentidos y tipos de memoria.

Seguidamente, se proporcionarán explicaciones concisas de los métodos clave presentes en la clase Memory, subrayando su funcionalidad:

```
class Memory:
    def __init__(self, factor):
        self.factor = factor
        self.life_history = {}
        self.life_episode = {}
        self.stats = {}
```

- `__init__(self, factor):`

  El constructor de la clase Memory inicializa una nueva instancia de memoria con un atributo `factor` proporcionado. Crea los diccionarios `life_history`, `life_episode` y `stats` para almacenar información sobre los eventos, episodios de vida y estadísticas de la memoria.

```
def add_memory(self, event, sense, neuron_number, pattern_list=None):
    existing_memory = self.life_history.get(event)

    if pattern_list is not None and existing_memory is not None:
        new_patterns = set(pattern_list) - set(existing_memory.get('pattern_list', []))
        existing_memory.setdefault('pattern_list', []).extend(new_patterns)
        existing_memory['neuron_number'] = neuron_number

    self.stats.setdefault(sense, {}).setdefault('number_registers', 0)
    self.stats[sense]['number_registers'] += 1

    if existing_memory is None:
        self.life_history[event] = {
            'event': event,
            'sense': sense,
            'neuron_number': neuron_number,
            'pattern_list': list(set(pattern_list)) if pattern_list is not None else None
        }
```

- `add_memory(self, event, sense, neuron_number, pattern_list=None):`

  Este método cumple con la tarea de integrar nuevos recuerdos al registro de la Historia de Vida (`life_history`). Si el recuerdo ya está presente en este registro, este método actualiza su lista de patrones, asegurándose de que no haya duplicados, y registra el número de neurona correspondiente. En situaciones en las que no hay recuerdo previo para el evento en cuestión, se crea una nueva entrada en la Historia de Vida con la información pertinente. Además de administrar los recuerdos, este método también rastrea estadísticas relacionadas con el sentido del evento, permitiendo un seguimiento detallado de las interacciones.

```
def fill_life_episode(self, event, sense, neuron_number, pattern_list):
    life_episode = {
        'event': event,
        'sense': sense,
        'neuron_number': neuron_number,
        'pattern_list': pattern_list
    }

    if sense not in self.stats:
        self.stats[sense] = {'number_registers': 0, 'number_occurrences': 0}

    self.stats[sense]['number_occurrences'] = 0
    self.life_episode[event] = life_episode
    seq = self.get_memory_sequence(sense)
    self.stats[sense]['number_occurrences'] = len(seq.split(','))
```

- `fill_life_episode(self, event, sense, neuron_number, pattern_list):`

  Se encarga de registrar un evento en la memoria episódica (`life_episode`) junto con la información relevante. Este método recibe varios parámetros: `event`, que es el identificador del evento; `sense`, que representa el sentido asociado al evento; `neuron_number`, que indica el número de neurona relacionado con el evento; y `pattern_list`, una lista de patrones asociados con el evento.

  Primero, se crea un diccionario `life_episode` que contiene los detalles del evento, como su identificador, sentido, número de neurona y la lista de patrones.

  Luego, se verifica si el sentido (`sense`) ya está registrado en las estadísticas (`stats`). Si no lo está, se crea una entrada en las estadísticas para ese sentido, inicializando el número de registros y el número de ocurrencias a cero.

  A continuación, se asigna el valor cero al número de ocurrencias para ese sentido en las estadísticas. Después, se agrega el evento y sus detalles al diccionario de memoria episódica (`life_episode`). Después de registrar el evento, se calcula la secuencia de memoria para el sentido dado utilizando el método `get_memory_sequence()`. Esta secuencia se almacena en la variable seq. Luego, se actualiza el número de ocurrencias para ese sentido en las estadísticas con la longitud de la secuencia de memoria calculada.


```
def update_memory(self, event, pattern):
    self.life_history[event]['pattern_list'] = [pattern]
```

- `update_memory(self, event, pattern):`

  Actualiza la lista de patrones asociados a un evento en la memoria. Es útil para establecer nuevas relaciones entre eventos y patrones.

```
def handle_attention(self, memory_sequence, pattern=None):
    new_event_key = memory_sequence.split(',')[0]
    new_event = self.life_episode.get(new_event_key, None)

    if self.life_history.get(new_event_key, None) is None and new_event is not None:
        self.add_memory(**new_event)

        if pattern is not None:
            self.update_memory(event=new_event['event'], pattern=pattern)

    elif self.life_history.get(new_event_key) is not None and pattern is not None:
        self.update_memory(event=new_event_key, pattern=pattern)

    self.life_episode = {}
```

- `handle_attention(self, memory_sequence, pattern=None):`

  Administra la atención en relación con un evento particular dentro de la secuencia de memoria proporcionada por el módulo de atención. Toma como entrada una cadena llamada `memory_sequence`, que representa una secuencia de eventos. A partir de esta secuencia, se identifica el primer evento clave en la secuencia.

  Primero, se extrae el identificador del nuevo evento clave de la cadena `memory_sequence`. Luego, se busca en la memoria episódica (`life_episode`) para obtener la información asociada con este nuevo evento clave.

  Si el evento clave no está presente en la memoria a largo plazo (`life_history`), significa que es un evento nuevo. En este caso, se agrega a la Historia de Vida utilizando los detalles del evento recuperados de la memoria episódica (`life_episode`).

  Por otro lado, si el evento clave ya existe en la Historia de Vida, se verifica si se proporcionó un patrón. Si es así, se actualiza el patrón del evento existente con el nuevo patrón.

  Finalmente, independientemente de si se creó o actualizó un evento, se restablece la memoria episódica (`life_episode`) a un estado vacío.

```
def get_memory_sequence(self, sense):
    event = next((value for value in self.life_episode.values() if value['sense'] == sense), None)
    memory_sequence = []

    if event:
        aux_event = self.life_history.get(event['event'])
        if aux_event is not None:
            combined_patterns = set(event['pattern_list']) | set(aux_event['pattern_list'])
            event['pattern_list'] = list(combined_patterns)

        element = event

        self._traverse_memory(element, memory_sequence, set())

    return ','.join(memory_sequence)

def __traverse_memory(self, memory: dict, memory_sequence: list, visited: set):
    if memory is None or memory['event'] in visited:
        return

    visited.add(memory['event'])
    memory_sequence.append(memory['event'])

    if memory['pattern_list'] is None:
        return

    for pattern in memory['pattern_list']:
        if pattern in self.life_history:
            self._traverse_memory(self.life_history[pattern], memory_sequence, visited)
```


- `get_memory_sequence(self, sense):`

  Este método tiene como objetivo construir una secuencia de recuerdos en función de un sentido específico. Comienza buscando el primer evento en el episodio de vida que esté asociado con el sentido proporcionado. Este evento se almacena en la variable `event`.

  Luego, se procede a construir la secuencia de recuerdos utilizando el método privado `__traverse_memory()`. Este método es responsable de recorrer y explorar la estructura de la memoria de manera recursiva, siguiendo las conexiones entre eventos y sus patrones relacionados. Dentro de `__traverse_memory()`, se verifica si el evento actual ya ha sido visitado para evitar bucles infinitos, manteniendo un registro en un conjunto llamado `visited`. Si el evento no ha sido visitado, se agrega a la secuencia de recuerdos `memory_sequence` y se procede a explorar sus patrones relacionados.

  Si un evento tiene una lista de patrones, se recorre cada patrón en esta lista. Si el patrón está presente en la historia de vida, se llama nuevamente a `__traverse_memory` para explorar los

recuerdos asociados con ese patrón, creando así una exploración en cascada de eventos y patrones. De vuelta en el método `get_memory_sequence()`, después de la exploración completa, los patrones relacionados con el evento original (`event`) se combinan con los patrones del evento auxiliar (`aux_event`), si es que existe. Esto asegura que la secuencia resultante incluya una visión completa y única de los recuerdos relacionados con el sentido dado.

Finalmente, la secuencia de recuerdos se devuelve como una cadena, donde los eventos están separados por comas, proporcionando una visión secuencial de cómo los eventos y sus patrones se relacionan en la memoria artificial.

4.9.2 Clase History

La clase History desempeña un papel fundamental en la gestión y coordinación de instancias de la clase Memory, permitiendo la administración integral de la historia de eventos y recuerdos. A continuación, se presentan los atributos y métodos de esta clase, detallando su contribución al funcionamiento de la arquitectura de memoria artificial.

La clase History se caracteriza por los siguientes atributos:

- `memories`: Un diccionario que almacena instancias de la clase Memory para diferentes tipos de memoria, como biológica, emocional y cultural. Estas instancias permiten la interacción y manipulación de los recuerdos y eventos asociados a cada tipo de memoria.

La clase History cuenta con una serie de métodos que facilitan la gestión y coordinación de operaciones relacionadas con las instancias de la clase Memory.

```
class History:
    def __init__(self):
        self.memories = {
            'biological': Memory('biological'),
            'emotional': Memory('emotional'),
            'cultural': Memory('cultural')
        }
```

- `__init__(self):`

  El constructor de la clase History se encarga de inicializar una nueva instancia de esta clase. Dentro del método, se crea el atributo `memories`, el cual se detalló previamente Este enfoque de creación y organización de instancias de memoria en la clase History permite manejar y acceder a diferentes tipos de memoria de manera más estructurada y eficiente dentro del sistema.

```
def init_history(self, arr_patterns):
    for sense_patterns in arr_patterns:
        for pattern in sense_patterns:
            self.add_pattern(pattern[0], pattern[1],
pattern[2])

def init_history_events(self, arr_events):
    for sense_patterns in arr_events:
        for value in sense_patterns:
            id_neuron = value[0]
            sense = value[4]
            event = value[3]
            pattern = [value[2].rstrip("_bce")]
            self.add_memory(event, sense, id_neuron, pattern)
```


- `init_history(self, arr_patterns) y init_history_events(self, arr_events):`

  Estos dos métodos se utilizan para la inicialización de la memoria con el propósito de precargar recuerdos. Son parte del proceso de preparación de la memoria para su uso.

  El primer método, `init_history()`, recibe una lista `arr_patterns` que contiene patrones junto con sus correspondientes atributos. Se itera sobre cada conjunto de patrones por sentido en `arr_patterns`. Luego, para cada patrón en el conjunto de patrones del sentido, se llama al método `add_pattern()` para agregar el patrón a la memoria con la información suministrada.

  El segundo método, `init_history_events()`, recibe una lista `arr_events` que contiene eventos junto con sus atributos. Al igual que en el método anterior, se itera sobre los conjuntos de eventos por sentido en `arr_events`. Para cada evento en el conjunto de eventos del sentido, se extraen los atributos relevantes, como el identificador de la neurona, el sentido, el evento y el patrón.

Luego se llama al método `add_memory()` para agregar el evento a la memoria con los atributos correspondientes.

En conjunto, estos métodos son esenciales para cargar previamente patrones y eventos en la memoria, estableciendo así una base de conocimiento inicial para el sistema.

```
def get_events(self, arr_events):
    for value in arr_events:
        id_neuron = value[0]
        sense = value[4]
        event = value[3]
        pattern = [value[2].rstrip("_bce")]
        self.fill_life_episode(event, sense, id_neuron, pattern)
```

- `get_events(self, arr_events):`

Procesa una lista de eventos que han sido recién generados por el agente inteligente. Cada evento contiene información como el identificador de neurona, el sentido, el evento y la lista de eventos asociados. El método extrae estos atributos y utiliza el método `fill_life_episode()` para registrar el evento en el episodio de vida correspondiente en la memoria. Esto contribuye a la construcción de la historia y el conocimiento en el sistema.


```
def add_memory(self, event, sense, neuron_number, pattern_list=None):
    for memory_type, memory_instance in self.memories.items():
        memory_instance.add_memory(event, sense, neuron_number, pattern_list)
```

- `add_memory(self, event, sense, neuron_number, pattern_list=None):`

Se encarga de agregar un nuevo recuerdo a todas las instancias de memoria almacenadas en el atributo `memories` de la clase History. Toma como entrada la descripción de un evento, el sentido asociado, el número de neurona y una lista de patrones que es opcional.

Para cada tipo de memoria almacenado en el atributo `memories`, el método itera a través de las instancias de memoria y llama al método `add_memory` de cada una. Esto permite que el recuerdo se agregue a todas las instancias de memoria correspondientes a los diferentes tipos (biológico, emocional, cultural)

```
def add_pattern(self, event, sense, neuron_number, pattern_list=None):
    memory_type = self.get_memory_type(event)
    if memory_type is None:
        raise ValueError("Invalid pattern.")

    memory = self.memories[memory_type]
    event_without_suffix = event.rstrip('_bce')
    memory.add_memory(event_without_suffix, sense, neuron_number,
pattern_list)
@staticmethod
def get_memory_type(event):
    suffix_map = {
        '_b': 'biological',
        '_e': 'emotional',
        '_c': 'cultural'
    }

    for suffix, memory_type in suffix_map.items():
        if event.endswith(suffix):
            return memory_type

    return None
```


- `add_pattern(self, event, sense, neuron_number, pattern_list=None):`

Es responsable de agregar un nuevo patrón a la memoria. Toma como entrada la descripción de un evento, el sentido asociado, el número de neurona y una lista de patrones que es opcional.

Primero, se determina el tipo de memoria al que pertenece el evento utilizando el método estático `get_memory_type()`. Si el tipo de memoria no se encuentra en el mapeo de sufijos, se lanza un error ya que el patrón es inválido.

Luego, se obtiene la memoria correspondiente del diccionario `memories` utilizando el tipo de memoria previamente identificado. El evento se procesa para eliminar cualquier sufijo "_bce" que pueda tener, y luego se utiliza el método `add_memory()` de la memoria seleccionada para

registrar este patrón en la historia, junto con el sentido y la lista de patrones, si dicha lista está presente.

El método estático `get_memory_type()` se encarga de mapear un evento a su tipo de memoria correspondiente. Utiliza un diccionario `suffix_map` que asocia sufijos de eventos con los tipos de memoria respectivos. Itera sobre este diccionario para verificar si el evento termina con alguno de los sufijos definidos. Si se encuentra un sufijo válido, devuelve el tipo de memoria correspondiente; de lo contrario, devuelve `None`, indicando que el evento no se puede asignar a un tipo de memoria conocido.

```
def get_memory_sequences(self, params: dict):
    memory_sequence_by_sense = {}

    for sense, sense_params in params.items():
        memory_sequence = {}
        for memory_type, memory_instance in self.memories.items():
            if sense_params[memory_type] == 1:
                memory_sequence[memory_type] = memory_instance.get_memory_sequence(sense)

        memory_sequence_by_sense[sense] = memory_sequence

    return memory_sequence_by_sense
```


- `get_memory_sequences(self, params: dict):`

Obtiene las secuencias de memoria correspondientes a diferentes sentidos y tipos de memoria. Toma como entrada un diccionario `params` que contiene información sobre qué tipos de memoria se deben considerar para cada sentido.

Dentro del método, se crea un diccionario `memory_sequence_by_sense` para almacenar las secuencias de memoria para cada sentido. Luego, se itera a través de los sentidos y sus respectivos parámetros en `params.`

Para cada sentido, el método recopila las secuencias de memoria de los tipos de memoria especificados en los parámetros. Utiliza la instancia de memoria correspondiente para obtener estas secuencias, utilizando el método `get_memory_sequence()` definido en la clase Memory.

```
def fill_life_episode(self, event, sense, neuron_number, pattern_list=None):
    for memory_type, memory_instance in self.memories.items():
        memory_instance.fill_life_episode(event, sense, neuron_number, pattern_list)
```

- `fill_life_episode (self, event, sense, neuron_number, pattern_list=None):`

  Desencadena el proceso de llenado de episodios de vida en varios tipos de memoria almacenados en instancias de la clase Memory. Este método se relaciona con el método `fill_life_episode()` definido en la clase Memory, que se encarga de almacenar información detallada sobre un evento en la memoria específica correspondiente.

  Cuando se llama al método `fill_life_episode()` en la clase History, se pasa información sobre el evento, el sentido, el número de neurona y, opcionalmente, una lista de patrones. Luego, el método itera a través de los diferentes tipos de memoria almacenados en `memories` (que son instancias de la clase Memory). Para cada tipo de memoria, se invoca el método `fill_life_episode()` de la instancia de memoria correspondiente, proporcionando la misma información del evento.


```
def handle_attention(self, factor, memory_sequence, pattern=None):
    memory_instance = self.memories.get(factor)
    memory_instance.handle_attention(memory_sequence, pattern)
```

- `handle_attention(self, factor, memory_sequence, pattern=None):`

  El método `handle_attention` en la clase History está relacionado con la definición previa en la clase Memory y se encarga de manejar la atención sobre un evento específico en la secuencia de memoria que ha sido procesada por el módulo de atención.

  Para realizar esto, el método recibe un parámetro `factor` que indica el tipo de memoria que se debe utilizar. Luego, utiliza este factor para obtener la instancia de memoria correspondiente almacenada en `memories`. Una vez obtenida la instancia de memoria, se invoca el método

`handle_attention` de esa instancia, pasándole la secuencia de memoria y, opcionalmente, un patrón.

En esencia, este método actúa como una interfaz para llamar al método `handle_attention` definido en la clase Memory específica para el tipo de memoria indicado por el factor. De esta manera, la clase History coordina el manejo de la atención en diferentes tipos de memoria almacenados en instancias de la clase Memory.

```
def get_stats(self):
    stats = defaultdict(lambda: defaultdict(lambda: {'number_registers': 0, 'number_occurrences': 0}))
    all_memory_types = set(self.memories.keys())

    for memory_type, memory_instance in self.memories.items():
        memory_stats = memory_instance.get_stats()
        for sense, sense_stats in memory_stats.items():
            stats[sense][memory_type].update(sense_stats)

    for sense_stats in stats.values():
        for memory_type in all_memory_types:
            sense_stats.setdefault(memory_type, {'number_registers': 0, 'number_occurrences': 0})

    stats_json = json.dumps(stats)
    stats_dict = json.loads(stats_json)

    return stats_dict
```


- `get_stats(self):`

Recopila y presenta estadísticas detalladas sobre los eventos y recuerdos presentes en las instancias de Memory. Organiza la información por sentido y tipo de memoria para una visión completa del impacto y la ocurrencia de los eventos.

```
def get_life_episodes(self):
    life_episodes = {}

    for memory_type, memory_instance in self.memories.items():
        for event, episode in memory_instance.life_episode.items():
            sense = episode['sense']
            if sense not in life_episodes:
                life_episodes[sense] = episode

    return life episodes
```

- `get_life_episodes()`:

  Agrupa y organiza episodios de vida de diferentes memorias según su sentido en un diccionario llamado `life_episodes`. Esta función está relacionada con el método homónimo en la clase Memory, pero en lugar de una sola memoria, opera en múltiples memorias para proporcionar una visión general de los episodios de vida en función de los sentidos.

Estos métodos y atributos de la clase History desempeñan un papel esencial en la orquestación y administración de eventos y recuerdos en el marco de la simulación de memoria artificial.

4.9.3 Pruebas unitarias

Las pruebas unitarias son una parte fundamental del proceso de desarrollo de software que permite garantizar la funcionalidad correcta de las clases y métodos en un sistema. En este contexto, se han realizado un conjunto de pruebas unitarias para las clases clave en el proyecto. Estas clases son componentes esenciales del sistema de memoria y almacenamiento de eventos, y asegurarse de que funcionen de manera precisa y confiable es crucial para el correcto funcionamiento del proyecto.

A través de estas pruebas, se evaluaron cada uno de los métodos en las clases Memory y History, verificando su capacidad para gestionar eventos, episodios de vida, estadísticas y otras operaciones relacionadas con la memoria.

Las pruebas unitarias para la clase Memory son las siguientes:

```
def test_add_memory(self):
    self.memory.add_memory(event='event1', sense='sight', neuron_number=1)
    self.assertEqual(len(self.memory.life_history), 1)
    self.assertEqual(self.memory.stats['sight']['number_registers'], 1)
```

- `test_add_memory`: Agrega un evento de memoria usando el método `add_memory()` y verifica si el evento se agrega correctamente a `life_history` y si las estadísticas para el sentido se actualizan como se espera.

```
def test_fill_life_episode(self):
    self.memory.fill_life_episode(event='event1', sense='sight', neuron_number=1, pattern_list=['pattern1'])
    self.assertEqual(len(self.memory.life_episode), 1)
    self.assertEqual(self.memory.stats['sight']['number_occurrences'], 1)
```

- `test_fill_life_episode`: Llena un episodio de vida usando el método `fill_life_episode` y verifica si el episodio se agrega correctamente a `life_episode` y si las estadísticas para el sentido se actualizan como se espera.

```
def test_update_memory(self):
    self.memory.add_memory(event='event1', sense='sight', neuron_number=1)
    self.memory.update_memory(event='event1', pattern='pattern1')
    self.assertEqual(self.memory.life history['event1']['pattern list'], ['pattern1'])
```


- `test_update_memory`: Agrega un evento de memoria y luego actualiza su patrón usando el método `update_memory()`. Verifica si el patrón se actualiza correctamente en `life_history`.

```
def test_handle_attention_new_event(self):
    self.memory.add_memory(event='pattern1', sense='sight', neuron_number=1)
    self.memory.add_memory(event='event1', sense='sight', neuron_number=2, pattern_list=['pattern1'])
    self.memory.handle_attention(memory_sequence='event1,event2', pattern='new_pattern')
    self.assertEqual(len(self.memory.life_history), 2)
```

- `test_handle_attention_new_event`: Agrega dos eventos de memoria y luego simula el manejo de la atención con un evento nuevo. Verifica si el nuevo evento se agrega correctamente a `life_history`.

```
    def test_handle_attention_existing_event(self):
        self.memory.fill_life_episode(event='event1', sense='sight', neuron_number=1, pattern_list=['pattern1'])
        self.memory.add_memory(event='event2', sense='sight', neuron_number=2)
        self.memory.handle_attention(memory_sequence='event2', pattern='new_pattern')
        self.assertEqual(self.memory.life_history['event2']['pattern_list'], ['new_pattern'])
```

- `test_handle_attention_existing_event`: Llena un episodio de vida, agrega otro evento de memoria y simula el manejo de la atención con el evento existente. Verifica si el patrón del evento existente se actualiza correctamente.

```
def test_get_memory_sequence(self):
    self.memory.add_memory(event='pattern1', sense='sight', neuron_number=1)
    self.memory.add_memory(event='event1', sense='sight', neuron_number=2, pattern_list=['pattern1'])
    self.memory.fill_life_episode(event='event2', sense='sight', neuron_number=3, pattern_list=['event1'])
    sequence = self.memory.get memory sequence(sense='sight')
```

- `test_get_memory_sequence`: Agrega eventos de memoria y episodios de vida, luego recupera la secuencia de memoria usando el método `get_memory_sequence()`. Verifica si la secuencia recuperada coincide con la secuencia esperada.


```
def test_get_stats(self):
    self.memory.add_memory(event='pattern1', sense='sight', neuron_number=1)
    self.memory.add_memory(event='event1', sense='sight', neuron_number=2, pattern_list=['pattern1'])
    self.memory.fill_life_episode(event='event2', sense='sight', neuron_number=3, pattern_list=['event1'])
    stats = self.memory.get_stats()
    self.assertEqual(stats['sight']['number_registers'], 2)
    self.assertEqual(stats['sight']['number_occurrences'], 3)
```

- `test_get_stats`: Agrega eventos de memoria y episodios de vida, luego recupera las estadísticas usando el método `get_stats()`. Verifica si las estadísticas recuperadas coinciden con los valores esperados.

Continuando con las pruebas unitarias, se enfocará en evaluar los métodos de la clase History, que interactúa con y coordina el funcionamiento de las memorias en el sistema:

```
def test_init_history(self):
    arr_patterns = [
        [('pattern1_b', 'biological', 1), ('pattern2_e', 'emotional', 2)],
        [('pattern3_c', 'cultural', 3)]
    ]

    self.history.init_history(arr_patterns)
    self.assertIsNotNone(self.history.memories['biological'].life_history.get('pattern1'))
    self.assertIsNotNone(self.history.memories['emotional'].life_history.get('pattern2'))
    self.assertIsNotNone(self.history.memories['cultural'].life_history.get('pattern3'))
```

- `test_init_history`: Esta prueba verifica que el método `init_history()` de la clase History inicialice correctamente los patrones en las memorias correspondientes. Se proporciona una lista de patrones y se verifica que los patrones se almacenan adecuadamente en las memorias biológica, emocional y cultural.

```
def test_init_history_events(self):
    arr_events = [
        [(1, {}, 'pattern1', 'event1', 'sight')],
        [(2, {}, 'pattern2', 'event2', 'touch')],
        [(3, {}, 'pattern3', 'event3', 'taste')]
    ]

    self.history.init_history_events(arr_events)
    self.assertIsNotNone(self.history.memories['biological'].life_history.get('event1'))
    self.assertIsNotNone(self.history.memories['emotional'].life_history.get('event2'))
    self.assertIsNotNone(self.history.memories['cultural'].life_history.get('event3'))
```

- `test_init_history_events`: Comprueba que el método `init_history_events` de History inicialice eventos en las memorias adecuadas. Se proporciona una lista de eventos y se verifica que los eventos se almacenan correctamente en las memorias biológica, emocional y cultural.

```
    def test_get_events(self):
        arr_events = [
            [1, {}, 'pattern1', 'event1', 'sight'],
            [2, {}, 'pattern2', 'event2', 'touch']
        ]

        self.history.get_events(arr_events)
        self.assertEqual(len(self.history.memories['biological'].life_episode), 2)
        self.assertEqual(len(self.history.memories['emotional'].life_episode), 2)
        self.assertEqual(len(self.history.memories['cultural'].life_episode), 2)
```

- `test_get_events`: Asegura que el método `get_events()` de History procese eventos correctamente. Se proporciona una lista de eventos, se procesan y se verifica que los episodios de vida correspondientes se llenen adecuadamente en las memorias biológica, emocional y cultural.

```
def test_get_memory_sequence(self):
    self.history.add_memory(event='pattern1', sense='sight', neuron_number=1)
    self.history.add_memory(event='event1', sense='sight', neuron_number=2, pattern_list=['pattern1'])
    self.history.fill_life_episode(event='event2', sense='sight', neuron_number=3, pattern_list=['event1'])
    sequence = self.history.get_memory_sequences({'sight': {
        'biological': 1,
        'cultural': 1,
        'emotional': 1
    }})

    self.assertEqual(sequence['sight']['biological'], 'event2,event1,pattern1')
    self.assertEqual(sequence['sight']['emotional'], 'event2,event1,pattern1')
    self.assertEqual(sequence['sight']['cultural'], 'event2,event1,pattern1')
```

- `test_get_memory_sequence`: Evalúa que el método `get_memory_sequence()` de History recopile secuencias de memoria de acuerdo con los parámetros especificados. Se agregan eventos y patrones a las memorias, se llena un episodio de vida y se obtiene la secuencia de memoria para diferentes tipos de memoria y sentidos, verificando que sean correctas.

```
def test_handle_attention(self):
    self.history.add_memory(event='pattern1', sense='sight', neuron_number=1)
    self.history.add_memory(event='event1', sense='sight', neuron_number=2, pattern_list=['pattern1'])
    self.history.add_memory(event='event2', sense='sight', neuron_number=2, pattern_list=['event1'])
    self.history.fill_life_episode(event='event3', sense='sight', neuron_number=3, pattern_list=['event2'])
    memory_sequence = "event3,event2,event1"
    pattern = "new_pattern"

    self.history.handle_attention('biological', memory_sequence, pattern)
    new_event_key = memory_sequence.split(',')[0]

    self.assertIsNotNone(self.history.memories['biological'].life_history.get(new_event_key))
    self.assertEqual(self.history.memories['biological'].life_history[new_event_key]['pattern_list'], [pattern])
```

- `test_handle_attention`: Evalúa cómo el método `handle_attention()` de la clase History se comporta en un contexto donde se han almacenado eventos y episodios de vida en las memorias biológica, emocional y cultural. Estos eventos están relacionados con el sentido de la vista ('sight'). Luego, se crea una secuencia de memoria llamada `memory_sequence`, que contiene varios eventos junto con un nuevo patrón llamado `new_pattern`.

  La evaluación se realiza en tres pasos, uno para cada tipo de memoria: biológica, cultural y emocional. En cada etapa, se verifica que el método `handle_attention()` maneje la atención adecuadamente y garantice la creación de un nuevo evento en la memoria correspondiente, que en este caso sería la memoria biológica, cultural y emocional, respectivamente. Esto asegura que el método `handle_attention()` funcione de manera efectiva en múltiples contextos de memoria, conservando la integridad de los eventos y patrones relacionados.

Capítulo 5
Conclusiones y Recomendaciones

A lo largo del proceso, se enfrentaron diversos desafíos que demandaron soluciones creativas y un enfoque meticuloso para alcanzar los resultados esperados. Se afrontó con decisión la complejidad inherente en el diseño y desarrollo de esta máquina inteligente, lo que permitió superar obstáculos esenciales y obtener avances significativos.

La incorporación de los algoritmos de atención y el desarrollo del módulo de red sensorial han sido fundamentales en la construcción de una memoria artificial versátil y capaz de procesar información relevante. Estos enfoques resultaron eficientes, abriendo nuevas oportunidades para un sistema de memoria adaptable, capaz de aprender y evolucionar con cada experiencia.

Es relevante resaltar que este proyecto marca apenas el comienzo de un inspirador camino de futuras investigaciones y desarrollos en el ámbito de las memorias artificiales, cuyo objetivo primordial es lograr emular y comprender algunas de las funciones del cerebro humano. Con el avance de la tecnología y un mayor entendimiento de la neurociencia computacional, se abren amplias oportunidades para enriquecer y expandir estas memorias artificiales de manera significativa.

Además, este trabajo sienta las bases para explorar de manera más compleja la interconexión entre diversos módulos, lo que permitiría una comprensión más profunda y una replicación más precisa de los procesos cognitivos humanos.

Este proyecto ha sido un viaje de exploración en el campo de las memorias artificiales. A lo largo de este proceso, se cumplieron estos objetivos:

1. Confrontación de la Complejidad: Se afrontaron y superaron diversos desafíos, demostrando la capacidad para encontrar soluciones creativas y aplicar un enfoque meticuloso en la construcción de la máquina inteligente.

2. Memoria Temporal como Episodio de Vida: Se logró desarrollar la arquitectura y organización de la Memoria Temporal, modelándola como un Episodio de Vida. Esta representación dinámica permitió capturar la naturaleza fluida y la interconexión de la información en un formato similar a las experiencias humanas.

3. Memoria de Largo Plazo como Historia de Vida: Se estableció con éxito la arquitectura y organización de la Memoria de Largo Plazo, denominada "Historia de Vida". Esta estructura proporcionó un mecanismo efectivo para almacenar y recuperar información a largo plazo, emulando el proceso de almacenamiento de recuerdos humanos.

Es importante reconocer que, a pesar de los esfuerzos realizados en el desarrollo de la memoria artificial, existen ciertas limitaciones en el alcance y funcionalidad del sistema. Estas limitaciones pueden incluir:

- Capacidad de almacenamiento: A medida que la cantidad de datos y recuerdos almacenados en la memoria aumenta, puede surgir un desafío en términos de capacidad de almacenamiento. La capacidad de la memoria artificial puede verse limitada por la cantidad de recursos disponibles y la eficiencia del algoritmo utilizado para la gestión de la memoria. Esto puede requerir enfoques adicionales, como técnicas de compresión o estrategias de gestión de memoria más sofisticadas, para abordar este desafío.

- Procesamiento en tiempo real: La capacidad de procesar y responder en tiempo real a eventos sensoriales puede ser un desafío en aplicaciones. La complejidad computacional y la cantidad de información que debe ser procesada pueden impactar en la velocidad de respuesta del sistema. Investigar y desarrollar técnicas de optimización y paralelización pueden ayudar a mejorar la capacidad de procesamiento en tiempo real de la memoria artificial.

- Generalización de patrones: La capacidad de la memoria para generalizar patrones y aprender de nuevos datos y experiencias puede ser limitada en ciertos contextos. La capacidad de adaptación y aprendizaje continuo puede requerir estrategias de entrenamiento adicionales y algoritmos de aprendizaje más sofisticados. Investigar enfoques de transferencia de aprendizaje y aprendizaje incremental puede ser una dirección prometedora para abordar esta limitación.

En cuanto al trabajo futuro, existen diversas áreas en las que se puede continuar investigando y mejorando la memoria artificial. Las recomendaciones planteadas son:

- Mejora de la eficiencia: Investigar y desarrollar técnicas de optimización y algoritmos más eficientes para el almacenamiento y recuperación de información en la memoria artificial puede permitir un uso más eficiente de los recursos y una mayor capacidad de procesamiento.

- Interacción con otros sistemas de inteligencia artificial: Investigar la integración de la memoria con otros sistemas de inteligencia artificial, como sistemas de aprendizaje automático o sistemas de procesamiento de lenguaje natural, puede permitir una colaboración y un intercambio de información más eficientes entre diferentes componentes.

- Actualmente la memoria temporal o Episodio de Vida contiene el evento que se está recibiendo, así como se conoce que lo hace nuestro cerebro, sin embargo, este módulo se podría profundizar aún más, conocer a fondo cómo funciona para poder emularlo de una manera más cercana a la realidad.

Bibliografía

Albarrán, M (2023). Diseño e Implementación del Movimiento Autónomo en Python – Módulo B: Ámbito y Aros de Atención. Trabajo de Grado, Universidad de Los Andes, Mérida, Venezuela.

Alsamhi, S. H., Almalki, F. A., Al-Dois, H., Othman, S. B., Hassan, J., Hawbani, A., … & Saleh, H. (2021). Machine Learning For Smart Environments In B5g Networks: Connectivity and Qos. Computational Intelligence and Neuroscience, (2021), 1-23. https://doi.org/10.1155/2021/6805151

Biloskurskyi, R. (2022). Agile Methodology Of Implementation Of Erp Information Systems. Scientific opinion: Economics and Management, 1(77). https://doi.org/10.32836/2521-666x/2022-77-12

Casas, P. F. i., Pi, X., Casanovas, J., Jové, J. (2013). Definition Of Virtual Reality Simulation Models Using Specification and Description Language Diagrams. Lecture Notes in Computer Science, 258-274. https://doi.org/10.1007/978-3-642-38911-5_15

Chang, T., Jo, S., Lu, W. (2011). Short-term Memory To Long-term Memory Transition In a Nanoscale Memristor. ACS Nano, 9(5), 7669-7676. https://doi.org/10.1021/nn202983n

Cawley, O., Wang, X. F., Richardson, I. (2013). Lean Software Development – What Exactly Are We Talking About? Lean Enterprise Software and Systems, 16-31. https://doi.org/10.1007/978-3-642-44930-7_2

Clarkson, L., Roodenrys, S., Miller, L. M., Hulme, C. (2016). The Phonological Neighbourhood Effect On Short-term Memory For Order. Memory, 3(25), 391-402. https://doi.org/10.1080/09658211.2016.1179330

Collier, R. W., O'Hare, G. M. P. (2009). Modeling and Programming By Commitment Rules In Agent Factory. Handbook of Research on Emerging Rule-Based Languages and Technologies, 393-421. https://doi.org/10.4018/978-1-60566-402-6.ch017

Dietz, A., Azzaro-Pantel, C., Pibouleau, L., Domenech, S. (2005). A Framework For Multiproduct Batch Plant Design With Environmental Consideration: Application To Protein Production. Ind. Eng. Chem. Res., 7(44), 2191-2206. https://doi.org/10.1021/ie049499m

Dittrich, M., Denkena, B., Boujnah, H., Uhlich, F. (2019). Autonomous Machining – Recent Advances In Process Planning and Control. Journal of Machine Engineering, 1(19), 28-37. https://doi.org/10.5604/01.3001.0013.0444

Eyzell, M. (2023) BrainCEMISID_4.0 Diseño e implementación en Python del Movimiento Autónomo-Módulo A: Núcleo Agente Inteligente, sentidos, Red Neuronal Sensorial. Trabajo de Grado, Universidad de los Andes, Mérida, Venezuela.

Farrell, S., Oberauer, K., Greaves, M. J., Pasiecznik, K. S., Lewandowsky, S., Jarrold, C. (2016). A Test Of Interference Versus Decay In Working Memory: Varying Distraction Within Lists In a Complex Span Task. Journal of Memory and Language, (90), 66-87. https://doi.org/10.1016/j.jml.2016.03.010

Fernández, A., Río, S. d., Lopez, V., Bawakid, A., Jesus, M. J. d., Benítez, J. M. & Herrera, F. (2014). Big Data With Cloud Computing: An Insight On the Computing Environment, Mapreduce, And Programming Frameworks. WIREs Data Mining Knowl Discov, 5(4), 380-409. https://doi.org/10.1002/widm.1134

Finstad, K., Bink, M. L., McDaniel, M. A., Einstein, G. O. (2006). Breaks and Task Switches In Prospective Memory. Appl. Cognit. Psychol., 5(20), 705-712. https://doi.org/10.1002/acp.1223

Fraser, G., Wotawa, F. (2007). Test-case Generation and Coverage Analysis For Nondeterministic Systems Using Model-checkers. International Conference on Software Engineering Advances (ICSEA 2007). https://doi.org/10.1109/icsea.2007.71

Furbach, U., Hölldobler, S., Ragni, M., Schon, C., Stolzenburg, F. (2019). Cognitive Reasoning: a Personal View. Künstl Intell, 3(33), 209-217. https://doi.org/10.1007/s13218-019-00603-3

Garner, A., Rowland, D., Hwang, S., Baumgaertel, K., Roth, B., Kentros, C. & Mayford, M. (2012). Generation Of a Synthetic Memory Trace. Science, 6075(335), 1513-1516. https://doi.org/10.1126/science.1214985

Gibson, B., Gregory, J. B., Robinson, P. G. (2005). The Intersection Between Systems Theory and Grounded Theory: The Emergence Of The Grounded Systems Observer. QSR, 2(1), 3-21. https://doi.org/10.18778/1733-8077.1.2.02

Gordon, P., Hendrick, R., Johnson, M. (2001). Memory Interference During Language Processing. Journal of Experimental Psychology: Learning, Memory, and Cogni, 6(27), 1411-1423. https://doi.org/10.1037/0278-7393.27.6.1411

Grootswagers, T., Wardle, S. G., Carlson, T. A. (2017). Decoding Dynamic Brain Patterns From Evoked Responses: a Tutorial On Multivariate Pattern Analysis Applied To Time Series Neuroimaging Data. Journal of Cognitive Neuroscience, 4(29), 677-697. https://doi.org/10.1162/jocn_a_01068

Gutierrez, J. (2015). On Fixpoint Logics and Equivalences For Processes With Restricted Nondeterminism. Journal of Logic and Computation, 4(28), 779-807. https://doi.org/10.1093/logcom/exv032

Hassabis, D., Maguire, E. A. (2009). The Construction System Of the Brain. Phil. Trans. R. Soc. B, 1521(364), 1263-1271. https://doi.org/10.1098/rstb.2008.0296

Hitch, G. J., Towse, J. N., Hutton, U. (2001). What Limits Children's Working Memory Span? Theoretical Accounts and Applications For Scholastic Development. Journal of Experimental Psychology: General, 2(130), 184-198. https://doi.org/10.1037/0096-3445.130.2.184

Iqbal, S., Allen, G. (2010). Aspect-oriented Modeling: Issues and Misconceptions. 2010 Fifth International Conference on Software Engineering Advances. https://doi.org/10.1109/icsea.2010.57

Isaka, S. (2023). Developmental Autonomous Behavior: An Ethological Perspective To Understanding Machines. IEEE Access, (11), 17375-17423. https://doi.org/10.1109/access.2023.3246840

Johannes, v. O., Christian, H., F., G. B., João, S. (2019). Continual Learning With Hypernetworks.. https://doi.org/10.48550/arxiv.1906.00695

Kempf, J., Bozga, M., Maler, O. (2013). As Soon As Probable: Optimal Scheduling Under Stochastic Uncertainty. Tools and Algorithms for the Construction and Analysis of Systems, 385-400. https://doi.org/10.1007/978-3-642-36742-7_27

Kersten, M., Murphy, G. (2015). Reducing Friction For Knowledge Workers With Task Context. AI Magazine, 2(36), 33-41. https://doi.org/10.1609/aimag.v36i2.2581

Kliegel, M., McDaniel, M. A., & Einstein, G. O. (2008). Prospective memory: Cognitive, neuroscience, developmental, and applied perspectives. Psychology Press.

Köpke, B., Nespoulous, J. (2006). Working Memory Performance In Expert and Novice Interpreters. INTP, 1(8), 1-23. https://doi.org/10.1075/intp.8.1.02kop

Kothari, C. R. (2004). Research Methodology: Methods and Techniques (2nd ed.). New Age International.

Leemans, S. J. J., Polyvyanyy, A. (2020). Stochastic-aware Conformance Checking: An Entropy-based Approach. Advanced Information Systems Engineering, 217-233. https://doi.org/10.1007/978-3-030-49435-3_14

Lépine, R., Bernardin, S., Barrouillet, P. (2005). Attention Switching and Working Memory Spans. European Journal of Cognitive Psychology, 3(17), 329-345. https://doi.org/10.1080/09541440440000014

Luo, L., Toubia, O. (2015). Improving Online Idea Generation Platforms and Customizing The Task Structure On The Basis Of Consumers' Domain-specific Knowledge. Journal of Marketing, 5(79), 100-114. https://doi.org/10.1509/jm.13.0212

Matthies, C. (2018). Scrum2kanban. Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials. https://doi.org/10.1145/3194779.3194784

Mourato, J., Ferreira, L. C. d. S., Sá, J. C., Silva, F., Dieguez, T., Tjahjono, B. (2020). Improving Internal Logistics Of a Bus Manufacturing Using The Lean Techniques. IJPPM, 7(70), 1930-1951. https://doi.org/10.1108/ijppm-06-2020-0327

Ovando-Tellez, M., Kenett, Y. N., Holling, H., Bernard, M., Belo, J., Beranger, B., … & Volle, E. (2021). Brain Connectivity-based Prediction Of Real-life Creativity Is Mediated By Semantic Memory Structure.. https://doi.org/10.1101/2021.07.28.453991

Pao, M. L. (1993). Term and Citation Retrieval: A Field Study. Information Processing & Management, 1(29), 95-112. https://doi.org/10.1016/0306-4573(93)90026-a

Rose, N. S., Myerson, J., Roediger, H. L., Hale, S. (2010). Similarities and Differences Between Working Memory And Long-term Memory: Evidence From The Levels-of-processing Span Task. Journal of Experimental Psychology: Learning, Memory, and Cogni, 2(36), 471-483. https://doi.org/10.1037/a0018405

Santo, S. d., Villegas, P., Burioni, R., Muñoz, M. A. (2018). Landau–ginzburg Theory Of Cortex Dynamics: Scale-free Avalanches Emerge At the Edge Of Synchronization. Proc. Natl. Acad. Sci. U.S.A., 7(115). https://doi.org/10.1073/pnas.1712989115

Schneider, D. W., Logan, G. D. (2005). Modeling Task Switching Without Switching Tasks: a Short-term Priming Account Of Explicitly Cued Performance. Journal of Experimental Psychology: General, 3(134), 343-367. https://doi.org/10.1037/0096-3445.134.3.343

Taewoon Kim, Michael Cochez, Vincent Francois-Lavet, Mark Neerincx, Piek Vossen (2022). A Machine With Human-Like Memory Systems. https://doi.org/10.48550/arXiv.2204.01611

Tan, K., Tay, A., Lee, T., Heng, C. Mining Multiple Comprehensible Classification Rules Using Genetic Programming. https://doi.org/10.1109/cec.2002.1004431

Thagard, P., Litt, A. (2001). Models Of Scientific Explanation. The Cambridge Handbook of Computational Psychology, 549-564. https://doi.org/10.1017/cbo9780511816772.024

Timme, N. M., Ito, S., Myroshnychenko, M., Nigam, S., Shimono, M., Yeh, F., … & Beggs, J. M. (2016). High-degree Neurons Feed Cortical Computations. PLoS Comput Biol, 8(4), e1000092. https://doi.org/10.1371/journal.pcbi.1000092

Tulving, E. (2002). Episodic memory: From mind to brain. Annual Review of Psychology, 53, 1-25.

Waguespack, L. J., Schiano, W. T. (2013). Thriving Systems Theory: An Emergent Information Systems Design Theory. 2013 46th Hawaii International Conference on System Sciences. https://doi.org/10.1109/hicss.2013.550

Wang, Y. (2011). Towards the Synergy Of Cognitive Informatics, Neural Informatics, Brain Informatics, And Cognitive Computing. International Journal of Cognitive Informatics and Natural Intelligence, 1(5), 75-93. https://doi.org/10.4018/jcini.2011010105

Ward, M., Zedan, H. (2016). The Formal Semantics Of Program Slicing For Nonterminating Computations. J Softw Evol Proc, 1(29), e1803. https://doi.org/10.1002/smr.1803

Werkhoven, P. J., Kester, L., Neerincx, M. A. (2018). Telling Autonomous Systems What To Do. Proceedings of the 36th European Conference on Cognitive Ergonomics. https://doi.org/10.1145/3232078.3232238

Woodman, G. F., Vogel, E. K., Luck, S. J. (2001). Visual Search Remains Efficient When Visual Working Memory Is Full. Psychol Sci, 3(12), 219-224. https://doi.org/10.1111/1467-9280.00339