Universidad de La Habana
Facultad de Matemática y Computación

Construcción Incremental de Bases de Conocimiento Semi-estructuradas con Modelos de Lenguaje

Autor: Carlos Mauricio Reyes Escudero

Tutor: Dr.C Alejandro Piad Morffis

Trabajo de Diploma presentado en opción al título de
Licenciado en Ciencia de la Computación

Junio de 2025

∀ aquellos que aún buscan saber

Agradecimientos

A tantas personas debo estar siquiera parado aquí hoy, ni hablar del tiempo para dedicarle a mis proyectos y aventuras, que no me alcanza el espacio para agradecer lo suficiente, y tampoco quiero que los profesores digan que hay más agradecimientos que tesis. Siento casi redundante agradecer a mi familia: si el universo es determinista, soy lo que soy por ellos, y si no lo es, lo soy gracias a ellos. Quiero agradecer a aquellos que han moldeado mi camino, poniendo a veces obstáculos y a veces dándome superpoderes; y a aquellos con quienes he tenido que saltar esos obstáculos o celebrar juntos las victorias. Perdonen la abstracción, pero pienso que en ella reside una gran habilidad del ser humano: saber de qué estamos hablando aunque solo dibujemos los bordes. Me deja pensando. Gracias.

Opinión del tutor

PLACEHOLDER PA LA OPINION DEL TUTOR

Resumen

La gestión del conocimiento personal (*PKM*) enfrenta el desafío de integrar eficientemente nueva información. Esta tesis aborda este problema mediante el desarrollo de un sistema que automatiza la construcción incremental de bases de conocimiento semiestructuradas utilizando Grandes Modelos de Lenguaje (*LLM*). Se presenta **lImporter**, un plugin para Obsidian.md implementado como un agente autónomo. La arquitectura del agente se basa en el paradigma *ReAct* (Razonamiento y Acción), permitiéndole planificar y ejecutar tareas complejas. Utiliza *function calling* para interactuar de forma fiable con la bóveda de notas del usuario y un sistema de optimización de contexto para analizar eficientemente el conocimiento existente y encontrar conexiones relevantes.

La eficacia y versatilidad del sistema se evaluaron a través de una serie de experimentos cualitativos. Estos demostraron la capacidad del agente para: (1) construir grafos de conocimiento estructurados a partir de texto no estructurado; (2) mantener y expandir una base de conocimiento existente, creando enlaces contextualmente apropiados; (3) adaptarse a flujos de trabajo de productividad como *P.A.R.A.*; y (4) utilizar la base de notas como una memoria externa para resolver tareas iterativas, mostrando su potencial como herramienta computacionalmente universal. El trabajo concluye que la integración de agentes *LLM* en sistemas de *PKM* transforma estas herramientas de repositorios pasivos a colaboradores cognitivos activos, capaces de estructurar, conectar y generar nuevo conocimiento de manera autónoma.

Palabras clave: Gestión del Conocimiento Personal, Modelos de Lenguaje, Obsidian, Agentes Autónomos, ReAct, Grafos de Conocimiento.

Abstract

Personal Knowledge Management (PKM) faces the challenge of efficiently integrating new information. This thesis addresses this problem by developing a system that automates the incremental construction of semi-structured knowledge bases using Large Language Models (LLMs). We introduce **llmporter**, an Obsidian.md plugin implemented as an autonomous agent. The agent's architecture is based on the *ReAct* (Reasoning and Acting) paradigm, enabling it to plan and execute complex tasks. It leverages *function calling* for reliable interaction with the user's note vault and employs a context optimization system to efficiently analyze existing knowledge and find relevant connections.

The system's effectiveness and versatility were evaluated through a series of qualitative experiments. These demonstrated the agent's ability to: (1) build structured knowledge graphs from unstructured text; (2) maintain and expand an existing knowledge base by creating contextually appropriate links; (3) adapt to productivity workflows such as P.A.R.A.; and (4) use the note vault as external memory to solve iterative tasks, showing its potential as a computationally universal tool. This work concludes that integrating LLM agents into PKM systems transforms these tools from passive repositories into active cognitive collaborators, capable of autonomously structuring, connecting, and generating new knowledge.

Keywords: Personal Knowledge Management, Large Language Models, Obsidian, Autonomous Agents, ReAct, Knowledge Graphs.

Índice general

Introducción

La sobrecarga de información en la actualidad plantea un desafío significativo para la productividad y el aprendizaje efectivo. En este contexto, la gestión del conocimiento personal emerge como una disciplina fundamental, no solo para académicos e investigadores, sino para cualquier individuo que busque optimizar su capacidad de aprendizaje y generación de ideas. Como señala Ahrens, *How to Take Smart Notes: One Simple Technique to Boost Writing, Learning and Thinking – for Students, Academics and Nonfiction Book Writers*, una gestión eficaz de las notas y el conocimiento adquirido no solo facilita la escritura y el estudio, sino que transforma la manera en que se interactúa con la información, convirtiéndola en un activo dinámico y generador de nuevas perspectivas. El desarrollo de un *segundo cerebro* (Forte, *Building a Second Brain*), un sistema externo confiable para almacenar y conectar ideas, libera recursos cognitivos, permitiendo un enfoque más profundo en el pensamiento crítico y la creatividad.

El potencial de los Grandes Modelos de Lenguaje (*LLM*) para revolucionar este campo es inmenso. Estas tecnologías ofrecen la posibilidad de automatizar y enriquecer la integración del conocimiento, asistiendo en la destilación de información, la identificación de conexiones y la generación de contenido relevante dentro de las bases de conocimiento personales. La automatización de estos procesos no solo promete un aumento en la eficiencia, sino también una democratización del acceso a metodologías avanzadas de gestión del conocimiento.

Motivación y Planteamiento del Problema

El presente trabajo de tesis se enmarca en la intersección de la Gestión del Conocimiento Personal (PKM) y los avances en Inteligencia Artificial, específicamente los *LLM*. La investigación busca abordar los desafíos inherentes a la integración eficiente y significativa de nuevo conocimiento en bases de conocimiento personales semiestructuradas.

El problema científico que se aborda es la optimización del proceso de integración de conocimiento en dichas bases. Tradicionalmente, este proceso es manual y consume tiempo, especialmente al incorporar información de diversas fuentes y formatos, y al establecer conexiones relevantes. Se busca explorar cómo los *LLM* pueden automatizar

y enriquecer esta tarea, facilitando la asimilación de información y la adaptación a diferentes paradigmas de toma de notas (e.g., *Zettelkasten*, notas conectadas, resúmenes progresivos). El objeto de estudio es, por tanto, el proceso de construcción y enriquecimiento automatizado de bases de conocimiento personal mediante LLM, centrándose el campo de acción en sistemas basados en lenguajes de marcado como *Markdown* y la representación del conocimiento mediante grafos, con especial atención a los *trabajadores del conocimiento*.

La investigación se guía por la pregunta: ¿En qué medida la aplicación de *LLMs*, a través de un *framework* de agentes personalizables, puede automatizar y optimizar la integración de conocimiento proveniente de diversas fuentes (predominantemente no estructuradas) en bases de conocimiento personales semiestructuradas (como las basadas en *Markdown* y grafos), y cómo se adapta esta automatización a diferentes paradigmas de toma de notas?

Objetivos

Para responder a esta pregunta, el *Objetivo General* es avanzar en la automatización de la construcción incremental y progresiva de bases de conocimiento personal, con un enfoque en la integración contextualizada de información mediante *LLM*. Los *Objetivos Específicos* son:

a. Desarrollar un *framework* flexible, implementado como un *plugin* para *Obsidian.md*, capaz de procesar datos en múltiples formatos (e.g., texto, *PDF*, web, audio) para su integración en bases de conocimiento personales existentes.
b. Diseñar e implementar un agente autónomo basado en el paradigma *ReAct* y dotado de distintas capacidades (resumen, extracción de entidades, generación de enlaces, creación de notas atómicas) para la integración de conocimiento.
c. Evaluar la eficacia y adaptabilidad del sistema para soportar diversos paradigmas de toma de notas, mediante estudios de caso que abarcan desde la construcción de grafos de conocimiento estructurados hasta la gestión de tareas de productividad personal, analizando la coherencia y utilidad del conocimiento integrado.

Contribución y Alcance

Este esfuerzo continúa la línea de investigación de trabajos como Fraga et al., «On the Automatic Generation of Knowledge Connections», pero se enfoca en el aprovechamiento de LLM para generar y enriquecer información directamente en el contexto de una base de conocimiento personal existente o en desarrollo. Hasta donde alcanza el conocimiento del autor, no existe una herramienta que combine integralmente *LLM* para la construcción progresiva y contextualizada de una base de conocimiento personal basada en notas

interconectadas, considerando la diversidad de fuentes y paradigmas. Para abordar esta brecha, se ha diseñado y desarrollado **lImporter**, un prototipo funcional para *Obsidian.md*. Este sistema se implementa como un agente autónomo que, guiado por instrucciones del usuario, procesa archivos, interactúa con un *LLM* para planificar y ejecutar la extracción, síntesis e integración de conocimiento en una base de notas (existente o nueva), generando nuevas notas, resúmenes, conexiones y metadatos. Este trabajo explora, a través de una serie de experimentos prácticos y teóricos, la viabilidad, eficacia y los desafíos de dicha automatización, buscando una contribución significativa a la *PKM* y a las aplicaciones prácticas de los *LLM*.

Capítulo 1

Estado del Arte

Este capítulo presenta una revisión de los fundamentos teóricos y tecnológicos que sustentan esta investigación. Se explora, en primer lugar, el campo de la Gestión del Conocimiento Personal (*PKM*), abarcando sus metodologías, las bases de conocimiento y los sistemas digitales que las implementan. A continuación, se profundiza en los Grandes Modelos de Lenguaje (*LLM*), detallando las técnicas clave para su aplicación, como el *prompting*, la optimización de contexto y las arquitecturas de agentes. Finalmente, se analiza el estado actual de la extracción de conocimiento y la integración de estas tecnologías en aplicaciones de toma de notas. Este análisis contextualiza el problema de investigación y permite identificar las brechas y oportunidades que motivan el desarrollo propuesto en esta tesis.

1.1. Gestión del Conocimiento Personal

La Gestión del Conocimiento Personal (*PKM*, por sus siglas en inglés) se define como el proceso mediante el cual un individuo recopila, clasifica, almacena, busca, recupera y comparte conocimiento en sus actividades diarias (Grundspenkis, «Agent Based Approach for Organization and Personal Knowledge Modelling»). Esta disciplina es particularmente relevante para los denominados *knowledge workers*, profesionales para quienes el conocimiento es su activo más valioso y que dedican una parte significativa de su tiempo a gestionar grandes cantidades de información. La PKM busca empoderar a estos individuos para que sean más efectivos en sus entornos personales y sociales.

1.1.1. Toma de notas

La toma de notas, lejos de ser un mero acto de transcripción, es una estrategia fundamental para mejorar el aprendizaje y la retención de información (Jansen et al., «An Integrative Review of the Cognitive Costs and Benefits of Note-Taking»). Al tomar notas,

el individuo no solo registra la esencia de la información, sino que también se involucra en un proceso activo de filtrado, organización y reestructuración del conocimiento.

Existen diversas metodologías para la toma de notas, que pueden clasificarse ampliamente en lineales y no lineales. Las metodologías lineales implican registrar la información en el orden en que se recibe. Un ejemplo común es denominado *outlining*, donde las notas y pensamientos se organizan de manera estructurada y lógica. Otra técnica lineal es el *sentence method*, que consiste en anotar cada tema como una oración corta y simple, ideal para lecciones de ritmo rápido.

Las metodologías no lineales, por otro lado, utilizan la organización espacial y los diagramas para ensamblar la información. Entre estas se encuentra el *charting*, útil para temas que pueden desglosarse en categorías. El *mapping*, como los mapas mentales, organiza las ideas en una estructura de árbol a partir de un punto central. Las Notas Cornell, desarrolladas por Walter Pauk (Pauk et al., *How to Study in College*), dividen la página en secciones para notas, pistas y un resumen. Las *guided notes* proporcionan un esquema predefinido con puntos clave faltantes que el estudiante completa. Más allá de estas técnicas tradicionales, han surgido sistemas más sofisticados. El método *Zettelkasten*, popularizado por Niklas Luhmann y descrito en detalle por Ahrens, *How to Take Smart Notes: One Simple Technique to Boost Writing, Learning and Thinking – for Students, Academics and Nonfiction Book Writers*, se basa en la creación de notas atómicas interconectadas, formando una red de conocimiento que fomenta la generación de ideas y la escritura prolífica. Este sistema, aunque con raíces históricas, ha ganado nueva relevancia en la era digital. Por otro lado, el método PARA *(Projects, Areas, Resources, Archives)*, propuesto por Forte, *Building a Second Brain*, ofrece un marco para organizar la información digital en función de su accionabilidad, facilitando la gestión de proyectos y responsabilidades a largo plazo.

1.1.2. Bases de Conocimiento Personal

Una Base de Conocimiento Personal (*PKB*) es una herramienta electrónica utilizada por un individuo para expresar, capturar y recuperar conocimiento personal. Se diferencia de una base de datos tradicional al contener material subjetivo y específico del propietario. El concepto de extender la memoria y las capacidades cognitivas del individuo mediante herramientas externas no es nuevo. Ya en 1945, Vannevar Bush imaginó el *Memex* (de *memory extension*), un dispositivo electromecánico en el que un individuo almacenaría todos sus libros, registros y comunicaciones, mecanizado de tal manera que pudiera ser consultado con gran velocidad y flexibilidad (Bush, *As We May Think*). Bush previó un futuro donde la sobrecarga de información requeriría nuevas formas de acceder y conectar el conocimiento acumulado.

La forma en que una *PKB* organiza el conocimiento se define por su modelo de datos. Estudios como los de Davies y colegas (Davies et al., «Building the Memex Sixty Years Later: Trends and Directions in Personal Knowledge Bases»; Davies, «Still Building the Memex») han analizado estos modelos en función de su marco estructural (cómo se interrelacionan los elementos), los elementos de conocimiento (las unidades básicas de información) y su esquema (el nivel de semántica formal). Un aspecto crucial destacado es la transclusión, la capacidad de ver el mismo elemento de conocimiento en múltiples contextos sin duplicación.

Grafos de Conocimiento Personal (PKG). Dentro de las PKB, los Grafos de Conocimiento Personal han ganado prominencia. Un PKG representa el conocimiento como una red de nodos interconectados, donde cada nodo es una pieza de información y las aristas representan las relaciones entre ellas, a menudo con visualizaciones gráficas que facilitan la exploración y el descubrimiento de conexiones (Pyne et al., «Meta-Work»).

1.1.3. Sistemas Digitales de Toma de Notas

La visión del *Memex* de Bush encuentra un eco contemporáneo en la plétora de sistemas digitales de toma de notas. Estos sistemas varían ampliamente en sus características y enfoques. Algunos, como *Roam Research* y *Logseq*, enfatizan los enlaces bidireccionales y la visualización gráfica, alineándose estrechamente con la idea de un *PKG*. Otros, como *Notion*, ofrecen una gran flexibilidad para crear bases de datos y vistas personalizadas, mientras que *Evernote* y *Google Keep* son populares por su simplicidad y accesibilidad multiplataforma. *Microsoft OneNote* tiene casi todos los *checks* en el articulo de la *Wikipedia*. Alternativas de código abierto como *Zettlr* y *Joplin* también ofrecen robustas funcionalidades, a menudo con un enfoque en la privacidad y el control local de los datos. Herramientas más tradicionales como *VimWiki* (para usuarios de *Vim*) y *OrgMode* (para *Emacs*) ofrecen sistemas de toma de notas altamente personalizables y potentes para usuarios con conocimientos técnicos. *Markor* y *SimpleNote* se centran en la simplicidad y la edición *Markdown*.

Entre estas herramientas, *Obsidian.md* ha ganado una considerable popularidad para la escritura académica y la gestión del conocimiento personal. Siguiendo el modelo de datos de las *PKB, Obsidian* almacena las notas como archivos locales de texto plano en formato *Markdown*, lo que garantiza la portabilidad y la longevidad de los datos. Su marco estructural se basa en un grafo de conocimiento, donde cada nota es un nodo y los enlaces bidireccionales permiten crear una red interconectada de información. Esto facilita la *transclusión*, ya que una misma idea o nota puede ser referenciada y contextualizada desde múltiples puntos del grafo sin necesidad de duplicación. Los elementos de conocimiento son flexibles, desde conceptos simples hasta notas extensas. Obsidian también soporta un

esquema enriquecido mediante el uso de metadatos (*frontmatter*), etiquetas y la posibilidad de extender su funcionalidad mediante *plugins*, como *Dataview*, que permite realizar consultas complejas sobre las notas. Su vista de grafo visualiza las conexiones, ayudando a identificar relaciones y patrones emergentes.

1.2. Grandes Modelos de Lenguaje

Los Grandes Modelos de Lenguaje (*LLM*) representan un avance transformador en el campo de la inteligencia artificial, con la capacidad de comprender, generar y manipular el lenguaje natural a niveles sin precedentes. Modelos como *GPT-4* de *OpenAI* (OpenAI et al., *GPT-4 Technical Report*) y *Gemini* de *Google* (Team et al., *Gemini*) han demostrado habilidades notables en una amplia gama de tareas lingüísticas. En paralelo, la comunidad de código abierto ha respondido con modelos competitivos como *DeepSeek* (DeepSeek-AI et al., *DeepSeek-V3 Technical Report*), *Qwen* (Bai et al., *Qwen Technical Report*) y *LLaMa* (Grattafiori et al., *The Llama 3 Herd of Models*), impulsando la innovación y la accesibilidad en este campo.

1.2.1. Técnicas de *Prompting*

Para interactuar eficazmente con los LLM, se han desarrollado un sinfín de técnicas de *prompting*. Inicialmente, se descubrió que proporcionar unos pocos ejemplos (*few-shot learning*) mejoraba significativamente los resultados, apelando a la capacidad de generalización de los modelos sin necesidad de reentrenamiento (Brown et al., «Language Models Are Few-Shot Learners»). Investigaciones posteriores han explorado qué partes de estos ejemplos son cruciales, sugiriendo que la estructura del texto o el formato pueden ser más importantes que la etiqueta correcta para que el modelo aprenda la tarea (Min et al., «Rethinking the Role of Demonstrations»).

Diversas estrategias se centran en guiar el comportamiento del modelo. El *role-prompting* (Kong et al., *Better Zero-Shot Reasoning with Role-Play Prompting*), que asigna un rol específico al LLM (e.g., actuar como un experto), y el *style prompting* (Lu et al., *Bounding the Capabilities of Large Language Models in Open Text Generation with Prompt Constraints*), que especifica el estilo deseado, permiten adaptar la respuesta a necesidades concretas. Para mejorar la calidad en tareas complejas, se ha aprovechado la capacidad de los LLM para inducirlos a generar pasos intermedios de razonamiento, un enfoque conocido como *cadena de pensamiento* (*Chain-of-Thought*) (Nye et al., *Show Your Work*; Wei et al., *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*). Este paradigma se ha extendido incluso a escenarios sin ejemplos (*zero-shot CoT*), donde se instruye al modelo a *pensar paso a paso* (Kojima et al., *Large Language Models Are Zero-Shot Reasoners*; Wang et al., *Plan-and-Solve Prompting*).

Otras técnicas se enfocan en la descomposición y la consistencia. El *complexity-based prompting* aprovecha que respuestas más complejas a menudo correlacionan con una mayor probabilidad de acierto (Fu et al., *Complexity-Based Prompting for Multi-Step Reasoning*), mientras que la descomposición de problemas (*least-to-most prompting*) resuelve sub-tareas más simples para integrar sus respuestas en la solución final (Zhou et al., *Least-to-Most Prompting Enables Complex Reasoning in Large Language Models*). La consistencia también se ha utilizado como criterio, generando múltiples respuestas y eligiendo la más frecuente (*self-consistency*) (Wang et al., *Self-Consistency Improves Chain of Thought Reasoning in Language Models*). Para problemas que requieren planificación, estructuras como el *Tree of Thoughts* y el *Graph of Thoughts* permiten al LLM explorar diferentes caminos de razonamiento (Yao et al., *Tree of Thoughts*; Besta et al., «Graph of Thoughts»).

Finalmente, se han desarrollado enfoques de auto-mejora y meta-razonamiento. Técnicas como *Self-Ask* incitan al LLM a generar y responder preguntas de seguimiento para clarificar la tarea (Press et al., *Measuring and Narrowing the Compositionality Gap in Language Models*), mientras que *Self-Refine* establece un marco iterativo donde el modelo critica y mejora sus propias respuestas (Madaan et al., *Self-Refine*). El *step-back prompting* instruye al modelo a resolver primero una versión más abstracta del problema para simplificar la tarea (Zheng et al., *Take a Step Back*). Otros enfoques exploran la inducción de un análisis previo del modo de proceder (*meta-reasoning*) (Gao et al., *Meta Reasoning for Large Language Models*), el uso de representaciones cognitivas como el *Sketch of Thought* (Aytes et al., *Sketch-of-Thought*) o la generación y evaluación iterativa de pasos de razonamiento (*Cumulative Reasoning*) (Zhang et al., *Cumulative Reasoning with Large Language Models*).

1.2.2. Optimización de Contexto

A pesar de que los *LLM* modernos poseen ventanas de contexto nominalmente muy grandes, su capacidad para utilizar eficazmente toda esa información es limitada, un fenómeno conocido como *lost in the middle* (Liu et al., «Lost in the Middle: How Language Models Use Long Contexts»). La optimización de contexto busca mitigar este problema presentando al modelo únicamente la información más relevante y densa para una tarea específica.

La principal estrategia para ello es la generación aumentada por recuperación (*Retrieval-Augmented Generation* o *RAG*), que combina la memoria paramétrica del *LLM* con una memoria no paramétrica externa (Lewis et al., *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*). En lugar de procesar un documento extenso, un sistema *RAG* primero recupera los fragmentos de texto más relevantes de una base de conocimientos (e.g., *Wikipedia*) y luego los proporciona como contexto al LLM para que genere la respuesta final. La expansión de la ventana de contexto en modelos como *Gemini 1.5* y *Qwen2.5-1M* ha hecho que el aprendizaje con muchos ejemplos (*many-shot in-context learning*) se vuelva

una estrategia viable y potente, donde se pueden incluir numerosos ejemplos relevantes directamente en el *prompt* (Team et al., *Gemini 1.5*; Yang et al., *Qwen2.5-1M Technical Report*; Agarwal et al., *Many-Shot In-Context Learning*).

La efectividad de *RAG* depende críticamente de la granularidad de la información recuperada. En lugar de recuperar pasajes de longitud fija, se ha demostrado que el uso de unidades más finas como las *proposiciones* —unidades de texto atómicas y autocontenidas— mejora la densidad de la información relevante, reduce el ruido y aumenta el rendimiento en tareas posteriores (Chen et al., «Dense X Retrieval»).

Más allá de la recuperación, han surgido marcos de compresión y razonamiento agentivo. **QwenLong-CPRS** es un sistema que, mediante instrucciones en lenguaje natural, comprime dinámicamente un contexto extenso en un resumen optimizado y específico para la consulta, actuando como un intermediario inteligente antes de pasar la información al *LLM* final (Shen et al., *QwenLong-CPRS*). Por otro lado, la *Cadena de Clarificaciones* (*Chain-of-Clarifications*) propone un flujo de trabajo en el que el modelo se enseña a sí mismo generando preguntas de clarificación sobre la consulta original, recuperando evidencia para responderlas y refinando iterativamente su comprensión antes de dar la respuesta definitiva (Zhuang et al., *Self-Taught Agentic Long Context Understanding*). Finalmente, los *Recitation-Augmented LLMs* se centran en mejorar la capacidad del modelo para recordar y citar fielmente la información del contexto proporcionado, ajustándolo para que aprenda a copiar explícitamente los segmentos relevantes, lo que mejora la fiabilidad de las respuestas basadas en el texto (Sun et al., *Recitation-Augmented Language Models*).

1.2.3. Agentes Basados en *LLM*

El verdadero potencial de los *LLMs* reside en su capacidad para actuar de forma autónoma y utilizar herramientas externas para superar sus limitaciones inherentes, como en cálculos matemáticos, razonamiento complejo o la verificación de hechos. A medida que los *LLM* han mejorado, investigadores y empresas han explorado cómo permitirles interactuar con sistemas externos.

El sistema *MRKL* (*Modular Reasoning, Knowledge, and Language*) (Karpas et al., *MRKL Systems*) es una de las formulaciones más simples de un agente, utilizando un LLM como *router* para acceder a múltiples herramientas (e.g., obtener el clima o la fecha actual) y combinar la información para generar una respuesta final. Modelos como *PAL (Program-aided Language Model)* (Gao et al., *PAL*) traducen problemas directamente a código ejecutable, mientras que *ToRA (Tool-Integrated Reasoning Agent)* (Gou et al., *ToRA*) intercala pasos de código y razonamiento. El paradigma *ReAct (Reasoning and Acting)* (Yao et al., *ReAct*) permite a los agentes generar un pensamiento, tomar una acción y recibir una observación, manteniendo un historial de estos pasos para informar decisiones futuras. *Reflexion* (Shinn et al., *Reflexion*) extiende *ReAct* incorporando retroalimentación lingüística para

refinar el comportamiento del agente. (*RAG*) se considera un sistema de agente cuando la propia recuperación se trata como una herramienta externa.

Para organizar la creciente diversidad de agentes y establecer un marco conceptual coherente, se han propuesto las *Arquitecturas Cognitivas para Agentes de Lenguaje* (*CoALA*, por sus siglas en inglés) (Sumers et al., *Cognitive Architectures for Language Agents*). Inspirado en la ciencia cognitiva y la inteligencia artificial simbólica, *CoALA* describe a los agentes de lenguaje en función de tres componentes clave: una memoria modular (de trabajo, episódica, semántica y procedimental), un espacio de acciones estructurado (acciones internas como razonar o aprender, y acciones externas para interactuar con el entorno) y un proceso de toma de decisiones generalizado. Este marco no solo permite clasificar y comparar los sistemas existentes, como *ReAct* o *Reflexion*, sino que también proporciona un plano para diseñar agentes futuros más capaces y estructurados, contextualizando los desarrollos actuales dentro de la historia más amplia de la IA.

1.3. Extracción de Conocimiento y Enlaces

La extracción de conocimiento y la creación de enlaces entre piezas de información son tareas fundamentales en la construcción de bases de conocimiento robustas y útiles. Tradicionalmente, estos procesos han sido manuales y laboriosos, pero los avances recientes en *LLMs* han abierto nuevas vías para su automatización y enriquecimiento.

1.3.1. Extracción de Conocimiento

La extracción de conocimiento, en el contexto de la gestión del conocimiento personal, implica identificar y capturar las ideas clave, entidades y relaciones presentes en diversas fuentes de información. Los *LLM* han demostrado una notable capacidad para esta tarea, abordándola desde diversas perspectivas metodológicas.

Un enfoque fundamental es la *extracción de conocimiento abierto*, que busca generar representaciones lógicas a partir de texto sin restricciones de dominio. Investigaciones pioneras como la de (Van Durme et al., «Open Knowledge Extraction through Compositional Language Processing») utilizan el procesamiento composicional del lenguaje para derivar proposiciones estructuradas, mientras que trabajos más recientes como (Song et al., «OpenFact») se centran en la veracidad y expresividad de los tripletes extraídos, utilizando marcos semánticos y enlaces a *Wikidata* para garantizar la calidad. A escala industrial, (Qian et al., *Open Domain Knowledge Extraction for Knowledge Graphs*) presenta un *framework* para extraer conocimiento de la web abierta, abordando desafíos de volumen, variedad y veracidad.

Para estructurar este proceso, se han propuesto diversos *frameworks* y *pipelines* basados en *LLM*. El trabajo de (Zhang et al., *Extract, Define, Canonicalize*) introduce un marco de

tres fases (Extraer-Definir-Canonizar) que realiza una extracción abierta seguida de una definición y canonización de esquemas *post-hoc*, permitiendo una gran flexibilidad. De manera similar, (Kommineni et al., *From Human Experts to Machines*) propone un *pipeline* semi-automatizado que abarca todo el proceso de construcción de grafos de conocimiento, desde la generación de preguntas de competencia (*CQs*) hasta la creación de la ontología y su poblamiento, utilizando un '*LLM* juez' para la evaluación. Un paradigma innovador es el de (Li et al., *KnowCoder*), que representa los esquemas de conocimiento como clases de Python, un formato que los *LLM* comprenden de forma nativa, y emplea un aprendizaje en dos fases para la comprensión y seguimiento del esquema. Por otro lado, (Luo et al., *OneKE*) diseña un sistema multi-agente (*Schema, Extraction, Reflection*) para guiar la extracción de conocimiento en diversos dominios y formatos de datos.

La aplicación de estas técnicas a dominios específicos y la extracción guiada es otra área clave. Para el dominio clínico, (Li et al., *Automated Clinical Data Extraction with Knowledge Conditioned LLMs*) utiliza *LLM* condicionados con conocimiento externo mediante aprendizaje en contexto (*in-context learning*) para extraer información de informes médicos. Para garantizar la coherencia con bases de conocimiento existentes, enfoques como los de (Feng et al., *Ontology-Grounded Automatic Knowledge Graph Construction by LLM under Wikidata Schema*) y (McCusker, *LOKE*) utilizan ontologías preexistentes (como la de *Wikidata*) para guiar y fundamentar el proceso de extracción del *LLM*. Finalmente, la interacción con el usuario se explora en (Abolhasani et al., *Leveraging LLM for Automated Ontology Extraction and Knowledge Graph Generation*), que presenta un *pipeline* interactivo guiado por un algoritmo adaptativo de Cadena de Pensamiento (*CoT*) para alinear la extracción de ontologías con los requisitos del usuario.

1.3.2. Creación de Enlaces

La creación de enlaces se refiere al establecimiento de conexiones significativas entre las piezas de conocimiento extraídas. Los *LLM* no solo extraen, sino que también pueden inferir y predecir estas relaciones.

Un paradigma dominante consiste en utilizar los *LLM* para la predicción de enlaces como una tarea de inferencia. En este sentido, (Shu et al., *Knowledge Graph Large Language Model (KG-LLM) for Link Prediction*) convierte la estructura del grafo en *prompts* de lenguaje natural para afinar *LLM* en la tarea de predicción de enlaces multi-salto. De forma similar, (He et al., *LinkGPT*) entrena un *LLM* de extremo a extremo que integra información estructural pareada mediante un ajuste de instrucciones en dos etapas. Para abordar la escalabilidad en grafos de gran tamaño, (Bi et al., *LPNL*) propone un *pipeline* de muestreo en dos etapas y una estrategia de 'divide y vencerás' para gestionar la sobrecarga de información en los *prompts*.

Una estrategia diferente consiste en la creación de enlaces sin supervisión o con pocos ejemplos. El trabajo de (Carta et al., *Iterative Zero-Shot LLM Prompting for Knowledge Graph Construction*) destaca en este aspecto, utilizando un *pipeline* de *prompting* iterativo en modo *zero-shot,* que no requiere ejemplos ni recursos externos para construir el grafo de conocimiento.

Finalmente, otra línea de investigación utiliza los *LLM* como enriquecedores de características para modelos existentes. En lugar de predecir el enlace directamente, (Park et al., *Enhancing Future Link Prediction in Quantum Computing Semantic Networks through LLM-Initiated Node Features*) emplea un *LLM* para generar descripciones ricas de los nodos, que luego se utilizan como características iniciales para mejorar el rendimiento de los modelos de Redes Neuronales de Grafo (*GNN*) en la predicción de enlaces, especialmente en escenarios de arranque en frío (*cold-start*).

La combinación de estas dos capacidades —extracción y enlace— tiene el potencial de transformar la construcción de bases de conocimiento. Trabajos como los de (Zhu et al., «LLMs for Knowledge Graph Construction and Reasoning») y (Machado et al., «{LLM Store}: Leveraging Large Language Models as Sources of {Wikidata}-Structured Knowledge») exploran las capacidades generales de los *LLM* para estas tareas, el primero concluyendo que son mejores asistentes de inferencia que extractores, y el segundo implementando un *LLM* como un 'almacén' de conocimiento dinámico que sintetiza respuestas estructuradas al momento de la consulta. La investigación continúa abordando desafíos como la interpretabilidad, la escalabilidad y la superación de las limitaciones inherentes de los *LLM* mediante arquitecturas de agentes y la integración de herramientas externas.

1.4. *LLM* en Aplicaciones de Toma de Notas

Los avances en los *LLMs* han catalizado una diversidad de herramientas para automatizar y enriquecer la toma de notas y la gestión del conocimiento personal (*PKM*). Estas varían desde aplicaciones web hasta *plugins* para plataformas consolidadas, enfocándose en distintos casos de uso.

Un área prominente es la *asistencia conversacional y la respuesta a preguntas contextualizadas*. En *Logseq*, *plugins* como *Logseq Copilot* permiten interactuar con la IA, indexando las notas del usuario para ofrecer respuestas basadas en el propio conocimiento. En *Obsidian.md*, herramientas como *BMO Chatbot for Obsidian* y *ChatGPT MD* ofrecen interfaces de chat versátiles, conectándose a servicios en la nube y *LLMs* locales, y permitiendo referenciar la nota actual o notas enlazadas en las conversaciones. *Caret Obsidian Plugin* extiende esta funcionalidad al *Canvas* para un chat no lineal. *Notion AI* también integra de forma nativa la capacidad de responder preguntas sobre el espacio de trabajo.

La *generación y mejora de contenido* es otro caso de uso fundamental. Herramientas como *LLM Summary* en *Obsidian* automatizan la creación de resúmenes de archivos *PDF* y la extracción de conceptos clave. Para la creación de material de estudio, *plugins* como *Obsidian Flashcards LLM* y *Quiz Generator* facilitan la generación de tarjetas de estudio y cuestionarios. En Logseq, *ollama-logseq* y *logseq-rag* permiten generar resúmenes y tarjetas de estudio. La mejora de la escritura y la reestructuración de notas se abordan con *Zettelkasten LLM Tools* y *Simple Prompt* en Obsidian, mientras que *LaTeX Generator Plugin for Obsidian* convierte lenguaje natural a ecuaciones *LaTeX*. Para la generación de contenido dentro de lienzos visuales, *Canvas LLM Extender* en Obsidian añade nodos de texto generados por IA.

En cuanto a la *organización, estructuración y enlace del conocimiento, InfraNodus* destaca como una aplicación web que visualiza texto como redes para identificar términos influyentes y lagunas conceptuales. En Obsidian, *ExMemo Tools* se enfoca en la gestión de documentos y la generación de metadatos como etiquetas, una tarea que *Obsidian LLM Tagger Plugin* también realiza usando *LLMs* locales. *InsightA Obsidian Plugin* transforma artículos largos en notas atómicas interconectadas y genera Mapas de Contenido (*MOCs*). *Obsidian Cloud Atlas Plugin* utiliza reconocimiento de entidades para crear *wikilinks* automáticamente, mejorando la interconexión.

La integración con *LLMs* locales para mayor privacidad y control es una tendencia creciente. En Logseq, *ollama-logseq* y *logseq-rag* se integran con *Ollama*. Para Obsidian, *Local LLM Helper - Obsidian Plugin* y *Obsidian AI plugin* conectan con servidores locales como *Ollama* o *LM Studio*. Muchas de las herramientas de chat y generación de contenido mencionadas también ofrecen la opción de operar con *LLMs* locales.

Finalmente, la *automatización de flujos de trabajo* y el *RAG* permiten interacciones más sofisticadas. *Logseq Composer* conecta notas con cualquier *LLM* mediante *RAG* a través de *LiteLLM*. En *Obsidian*, *Cannoli* permite construir *scripts* de *LLM* sin código en el editor *Canvas*, y *LLM Workspace plugin for Obsidian* permite crear conjuntos de fuentes curadas para fundamentar las conversaciones con IA mediante *RAG*. *Obsidian Cloud Atlas Plugin* también introduce flujos de trabajo en *Canvas* o *Markdown*.

Esta evolución subraya un esfuerzo por automatizar tareas, mejorar la interconexión de ideas y potenciar la utilidad de los sistemas de *PKM*, con una diversidad de enfoques que buscan adaptarse a las necesidades específicas de los usuarios.

Capítulo 2

lImporter

Este capítulo presenta la propuesta de *lImporter*, un plugin para *Obsidian* diseñado para automatizar la integración de nuevo conocimiento en una bóveda existente. El sistema recibe un conjunto de archivos y, guiado por instrucciones del usuario, genera y vincula nuevas notas de manera inteligente.

El problema fundamental que se aborda puede definirse formalmente. Se construye un Agente, A, que opera sobre una bóveda de *Obsidian*. La función del agente es transformar la bóveda a un nuevo estado basándose en un conjunto de archivos de entrada y unas instrucciones específicas.

Sea V el estado de la bóveda, definido como un conjunto de notas en formato *Markdown*. Sea $F = \{f_1, f_2, \ldots, f_n\}$ un conjunto de archivos de entrada y sea I un conjunto de instrucciones en lenguaje natural que guían el proceso. La operación del agente se puede describir como una función:

$$A(F, I, V) \rightarrow V'$$

donde V' es la bóveda actualizada, que contiene nuevas notas, modificaciones a notas existentes y nuevos vínculos entre ellas, como resultado de la evaluación del agente. El conjunto de archivos de entrada F puede estar compuesto por múltiples formatos. Para el reconocimiento de su contenido, se aprovecha la multimodalidad nativa de los modelos *Gemini* (Team et al., *Gemini*), que pueden procesar texto, imágenes y audio directamente. El componente de extracción de contenido puede ser reemplazado fácilmente por herramientas especializadas, como un sistema de reconocimiento óptico de caracteres (*OCR*) para archivos *PDF* o un servicio de *speech-to-text* para audio.

Para la implementación de *lImporter*, se propone un agente autónomo basado en el paradigma *ReAct (Reasoning and Acting)*. Este agente está dotado de un conjunto de herramientas especializadas que le permiten interactuar directamente con la bóveda de *Obsidian*, leyendo su estructura, obteniendo el contenido de las notas y escribiendo nueva información.

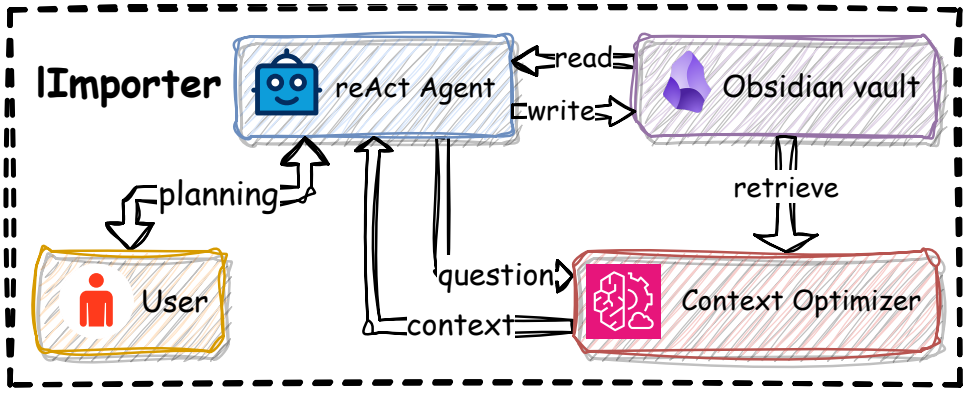


Figura 2.1. Esquema general del flujo de trabajo del agente en *lImporter*.

La explicación de la arquitectura se realizará de manera *top-down*. Se comenzará describiendo el entorno y el funcionamiento general del agente, para luego profundizar en cada uno de sus componentes clave: el mecanismo de interacción con la bóveda y las estrategias para la obtención de contexto relevante.

2.1. Descripción del entorno

El entorno de trabajo para este proyecto es *Obsidian*. A diferencia de otras herramientas basadas en la nube, el modelo de datos de *Obsidian* es simple y robusto: una carpeta local en el sistema de archivos del usuario que contiene todos los datos. Las notas se almacenan como archivos de texto plano en formato *Markdown* (‘*.md*‘), y pueden organizarse en una jerarquía de carpetas tradicional. Este enfoque garantiza la portabilidad, la longevidad y el control total del usuario sobre su información.

Una de las características más distintivas de *Obsidian* es su capacidad para visualizar las conexiones entre notas como un grafo de conocimiento. Cada nota es un nodo en el grafo, y los vínculos entre notas se representan como aristas. Esto permite al usuario descubrir relaciones emergentes y navegar por su conocimiento de una manera no lineal. El grafo de *Obsidian* es un **grafo dirigido por fuerzas** (*force-directed graph*), una simulación física donde los nodos se repelen entre sí, mientras que los enlaces actúan como resortes que los atraen. Este mecanismo provoca que los nodos con conexiones en común, incluso indirectas, se agrupen visualmente, revelando clústeres temáticos de forma intuitiva (ver Figura 2.3).

La creación de vínculos es fundamental para construir el grafo. En *Obsidian*, esto se logra mediante la sintaxis de *wikilinks*, simplemente encerrando el nombre de otra nota entre dobles corchetes, por ejemplo, ‘[[Nota Destino]]‘. Además de los vínculos simples, *Obsidian* soporta la **transclusión**, que permite incrustar el contenido de una nota (o una sección de ella) dentro de otra usando la sintaxis ‘![[Nota a Incrustar]]‘. En la estructura del

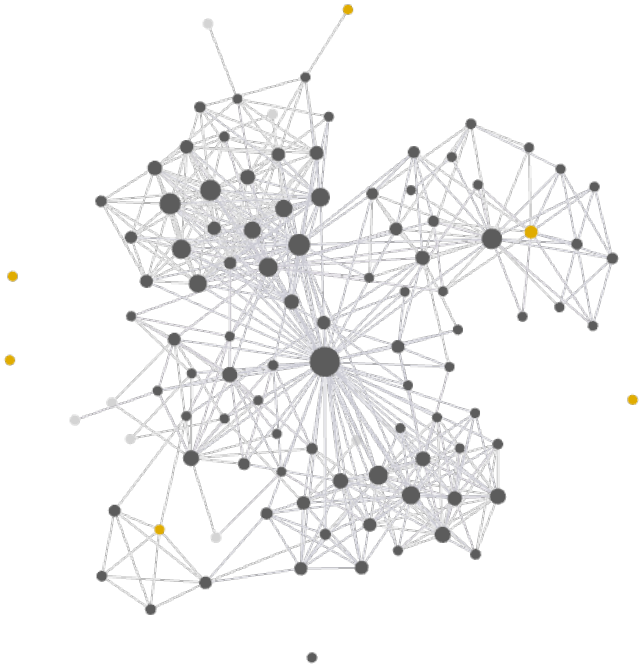


Figura 2.2. Ejemplo de la vista de grafo en *Obsidian,* mostrando nodos y sus interconexiones.

grafo, tanto un vínculo simple ‘[[...]]‘ como una transclusión ‘![[...]]‘ crean una conexión directa y equivalente entre los nodos correspondientes, contribuyendo por igual a la dinámica de fuerzas. La transclusión, por tanto, enriquece el contenido de la nota sin alterar la naturaleza del vínculo en el grafo.

2.2. Agente *reAct*

En el núcleo del sistema se encuentra un agente basado en el *framework ReAct* (Yao et al., *ReAct*). Este paradigma permite a los modelos de lenguaje combinar el razonamiento (*thought*) con la acción (*action*). El agente opera en un ciclo donde, a partir de una tarea y una observación del entorno, razona sobre el siguiente paso a seguir, elige una herramienta de su repertorio, la ejecuta, y observa el resultado para informar su siguiente ciclo de razonamiento. En la práctica, este ciclo de *Thought-Action-Observation* no es más que un bucle ‘while(true)‘ que se ejecuta hasta que se cumple una condición de parada (Figura 2.6). La sesión del agente finaliza de tres maneras: cuando el agente invoca una herramienta específica de finalización (e.g., `finish()`), cuando en su paso de razonamiento decide que la tarea está completa y no devuelve ninguna llamada a función, o cuando se alcanza un límite máximo de llamadas previamente definido.

Figura 2.3. Comparación de la vecindad de la etiqueta `#Large_Language_Models` y la nota `Large_Language_Models.md`.

La proximidad visual entre el nodo de la etiqueta (izquierda) y el nodo de la nota (derecha) es un efecto directo del diseño basado en fuerzas. Al estar ambos conectados a un conjunto casi idéntico de notas, son atraídos hacia sus vecinos comunes, lo que resulta en su agrupación natural dentro del grafo.

Siguiendo las mejores prácticas observadas en agentes avanzados como los utilizados para investigación o codificación (e.g. *Deep Research*), se ha adoptado la estrategia *Plan-and-Solve* (Wang et al., *Plan-and-Solve Prompting*). En lugar de actuar de forma impulsiva, el agente primero genera un plan detallado y paso a paso para resolver la tarea. El usuario tiene la opción de revisar, confirmar y dar retroalimentación sobre el plan generado antes de su ejecución. Este enfoque de *human-in-the-loop* combina la eficiencia del planeamiento autónomo con la supervisión y el direccionamiento estratégico del usuario.

Para la implementación del agente se utiliza la familia de modelos *Gemini* de *Google*, debido a su multi-modalidad nativa, accesibilidad y, de manera crucial, su soporte nativo para *function calling*, un mecanismo que se detallará a continuación.

2.2.1. Interacción con la bóveda

La capacidad del agente para leer y escribir archivos en la bóveda es posible gracias al uso de la *API* de *Obsidian*, orquestada a través de *Function Calling*. Esta técnica puede ser entendida como un caso particular de *style prompting* que emplea decodificación restringida (*constrained decoding*) para forzar al modelo de lenguaje a generar una salida que se adhiere estrictamente a una gramática o formato predefinido (Geng et al., *Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning*).

En el caso de la *API* de *Gemini*, esta restricción obliga al modelo a producir una salida en formato *JSON* que se corresponde con la firma de una de las funciones (herramientas) disponibles. Se aprovecha esta capacidad para mejorar la fiabilidad del agente. Por ejemplo, al definir los parámetros de una función, se puede utilizar el tipo de dato '*ENUM*' para restringir las entradas a un conjunto de valores válidos (e.g., una lista de carpetas o archivos existentes), evitando así que el modelo intente operar sobre rutas inválidas o alucinadas.

Lectura

Para la lectura de información de la bóveda, el agente dispone de dos herramientas principales:

- `tree(path)`: Esta función recibe la ruta a una carpeta raíz y devuelve una representación textual de la estructura de directorios y archivos contenida en ella, de forma análoga al comando `tree` de la línea de comandos. Esto permite al agente obtener una visión global de la organización de la bóveda.
- `read(path)`: Esta función recibe la ruta a un archivo *Markdown* ('*.md*') existente y devuelve su contenido completo como texto.

Escritura

Un desafío común en los agentes que interactúan con sistemas de archivos es la tendencia del modelo a generar rutas de archivo incorrectas o con nombres similares pero no idénticos a los existentes. Para mitigar este problema, la capacidad de escritura se ha separado en dos funciones distintas:

- `mkdir(path)`: Permite al agente crear un nuevo directorio en una ubicación específica de la bóveda, asegurando que la estructura de carpetas deseada exista antes de intentar escribir un archivo.
- `write(path, content)`: Crea un nuevo archivo o sobrescribe el contenido de uno existente. El parámetro `path` de esta función se beneficia directamente de la decodificación restringida, ya que se puede guiar al modelo para que elija entre rutas sugeridas o siga un formato válido, reduciendo drásticamente los errores.

Obtencion de contexto

Para que el agente pueda tomar decisiones informadas, como determinar dónde crear una nueva nota o con qué notas existentes vincularla, necesita un contexto relevante de la bóveda. Dado que el contenido total de la bóveda puede exceder fácilmente la ventana de contexto del modelo, se utiliza un enfoque de divide y vencerás.

Se propone una función que recibe un conjunto de archivos y un límite de tokens. Si el contenido combinado de los archivos excede el límite, el conjunto se divide recursivamente por la mitad hasta que los fragmentos resultantes son lo suficientemente pequeños. Una vez que un fragmento cabe en la ventana de contexto, un modelo de lenguaje se encarga de extraer la información relevante de él. La naturaleza de esta información relevante se define por un nivel de **granularidad** especificado por el usuario (en Chen et al., «Dense X Retrieval» se analiza la optimalidad de selección de diferentes casos de estos).

Esta idea está inspirada en la técnica propuesta en Shen et al., *QwenLong-CPRS*. De manera similar, se proponen tres niveles de granularidad:

- **Paragraph**: Ideal para obtener resúmenes y el sentido general de un documento.
- **Sentence**: Extremadamente útil para identificar afirmaciones específicas y recuperar relaciones entre conceptos.
- **Keyword**: Perfecto para la extracción de entidades nombradas y conceptos clave.

```
function getContext(files, token_limit, granularity):
    if total_tokens(files) <= token_limit:
        return extract_info(files, granularity)
    else:
        mid = floor(len(files) / 2)

        part1 = files[0:mid]
        part2 = files[mid:len(files)]

        context1 = getContext(part1, token_limit, granularity)
        context2 = getContext(part2, token_limit, granularity)

        return combine(context1, context2)
```

Figura 2.4. Pseudocódigo del proceso de obtención de contexto.

2.2.2. Sobre la versatibilidad del sistema

Un objetivo clave de *lImporter* es su versatilidad para integrarse en diversos flujos de trabajo. El agente debe ser útil tanto para un usuario que desea construir una base de conocimiento densamente interconectada, generando múltiples relaciones y resúmenes, como para otro que simplemente quiere archivar un nuevo concepto de forma aislada, sin crear ningún vínculo.

La arquitectura basada en un agente *ReAct* y el sistema de obtención de contexto granular sustentan esta flexibilidad. El agente puede recibir instrucciones para ser exhaustivo en la búsqueda de conexiones o, por el contrario, para limitarse a crear una nota y guardarla en una carpeta específica. Un ejemplo claro de esta flexibilidad es el proceso de

selección de la ubicación para las nuevas notas. El sistema no prefija una carpeta de destino. En su lugar, el agente decide la ruta óptima en tiempo de ejecución basándose en una combinación de factores: las instrucciones iniciales del usuario, y las directrices que puede encontrar dentro del contenido de los propios archivos de entrada. Se le puede instruir, por ejemplo, para que lea un archivo `_index.md` que guíe la organización de un subdirectorio. Esta libertad permite al usuario implementar tanto estructuras rígidas como flujos de trabajo donde las notas se crean de manera dispersa y emergente. Para garantizar la integridad de la bóveda, aunque el agente puede explorar cualquier ruta desde la raíz para tomar su decisión, al momento de ejecutar la acción de escritura, los parámetros de la función se validan. Como se mencionó, se le presenta al modelo una lista de las carpetas existentes en un formato '*ENUM*', lo que restringe su elección a una ruta válida y previene errores.

Sin embargo, esta versatilidad conlleva un incremento en la labor manual de configuración inicial, ya que el comportamiento del agente se define en gran medida a través de un archivo de instrucciones proporcionado por el usuario.

```
list_of_existing_files = Vault.get_all_file_paths()
list_of_existing_dirs = Vault.get_all_dir_paths()

dynamic_constraints = {
    "read":{
            "path": ENUM(list_of_existing_files)
        },
    "write":{
            "directory": ENUM(list_of_existing_dirs),
            "filename": REGEX("^.+(?i)\\.md$")
        }
    }
```

Figura 2.5. Flujo de decodificación restringida para la interacción con el sistema de archivos. El agente expresa una intención, el sistema traduce las rutas válidas en restricciones y la API del modelo genera una llamada a herramienta que está garantizada para ser ejecutable.

Es importante aclarar que, al momento de la implementación, la API de *Gemini* no soporta de forma nativa la validación de parámetros mediante expresiones regulares (*regex*) en la definición de herramientas. Sin embargo, en la práctica, el modelo rara vez comete errores en la generación de nombres de archivo con el formato adecuado (por ejemplo, omitiendo la extensión `.md`). En cualquier caso, esta validación se puede implementar de forma trivial como un paso de post-procesamiento: el sistema recibe la llamada a la función generada por el modelo y verifica la validez del nombre del archivo antes de ejecutar la acción de escritura en el disco.

```
function react_agent(task, files, tools, model, max_steps, auto_accept_plan=false):
    planning_prompt = create_planning_prompt(task, files, tools)
    plan_response = model.generate(planning_prompt, forced_tool="propose_plan")
    proposed_plan = extract_plan_from_response(plan_response)

    if not auto_accept_plan:
        final_plan = user_confirm_or_edit(proposed_plan)
        if not final_plan:
            return "Ejecución cancelada por el usuario."
    else:
        final_plan = proposed_plan

    prompt = create_execution_prompt(task, files, final_plan, tools)

    for i in 1 to max_steps:
        response = model.generate(prompt)
        action = extract_action(response)

        if not action or action.name == "finish":
            print("Tarea completada.")
            return extract_final_answer(response)

        tool_to_call = tools.get(action.name)
        observation = tool_to_call.execute(action.args)

        prompt += f"\nAction: {action.name}({action.args})"
        prompt += f"\nObservation: {observation}\nThought:"

    print("Límite de pasos alcanzado.")
    return "Tarea no completada."
```

Figura 2.6. Pseudocódigo del ciclo de ejecución del agente *ReAct*.

Cabe señalar que el pseudocódigo presentado es una representación conceptual y simplificada del ciclo *ReAct*. En la práctica, se usa una implementación más sofisticada del paradigma. En lugar de gestionar el flujo mediante la concatenación manual de prefijos de texto (como `Thought:` u `Observation:`), una técnica común en tareas de predicción de texto simple, la interacción se gestiona de forma más integrada a través de la API del modelo. Esto implica mantener un historial de conversación estructurado, donde cada llamada a una herramienta y su correspondiente resultado (la observación) se añaden como mensajes distintos en el diálogo, aprovechando directamente las capacidades nativas de *function calling*.

Capítulo 3

Experimentos

En este capítulo se presentan una serie de experimentos diseñados para evaluar el rendimiento y las capacidades del agente *lImporter*. Es importante aclarar que no se realizó un análisis holístico y cuantitativo del sistema. En su lugar, se optó por explorar distintos casos de uso que demuestran la versatilidad de la herramienta y ponen de manifiesto las interesantes posibilidades que surgen al integrar un agente de lenguaje autónomo en una bóveda de conocimiento de *Obsidian*.

3.1. Base estructurada sobre *Harry Potter*

En el primer experimento, se utilizó la versatilidad del agente para una tarea clásica: la creación de una base de conocimientos estructurada sobre el universo de *Harry Potter*. El objetivo era proporcionar al agente el *PDF* del primer libro y pedirle que generara notas para personajes, lugares y conceptos clave, vinculándolos entre sí. El agente generó exitosamente un total de 296 notas (entidades) y 671 enlaces entre ellas.

Es relevante mencionar que, por defecto, *Obsidian* no muestra etiquetas en las aristas del grafo, lo que dificulta la interpretación de la naturaleza de las relaciones. Para solventar esto, se utilizó un *plugin* llamado *Graph Link Types* de la comunidad que añade dicha funcionalidad. El grafo resultante, como se muestra en la Figura 3.1, fue creado exitosamente.

Es importante señalar que este experimento no fue diseñado con el objetivo de realizar una extracción de entidades exhaustiva y precisa. Su propósito principal era demostrar la capacidad del sistema para abordar un caso de uso estructurado con la configuración adecuada. Aun así, los resultados evidencian cómo el agente puede generar rápidamente una representación del conocimiento que es tanto visualmente atractiva como funcional, constituyendo un excelente punto de partida para el análisis preliminar de cualquier dominio.

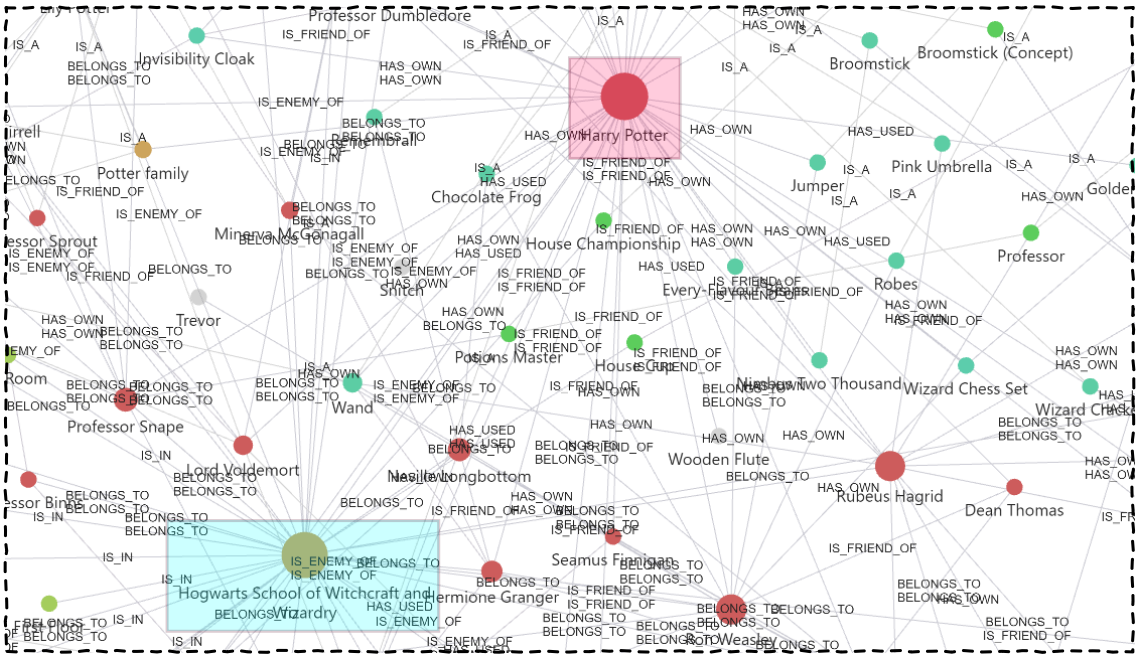


Figura 3.1. Grafo resultante del experimento de *Harry Potter*, mostrando la centralidad de nodos clave.

En la figura, se observa que conceptos centrales como *Harry Potter* (resaltado en rojo) y *Hogwarts* (en azul) se representan con nodos de mayor tamaño. Esto se debe a que *Obsidian* modula el tamaño de cada nodo en función de su grado —el número de conexiones que posee—, lo que resalta visualmente la importancia de las entidades más conectadas y demuestra la utilidad de esta representación incluso para bases de conocimiento ya estructuradas.

3.2. Añadiendo artículos a una *Wiki* tecnológica

Este experimento se diseñó para simular el mantenimiento y crecimiento de una base de conocimiento existente. Se partió de una bóveda de *Obsidian* que funcionaba como una *wiki* personal sobre temas de tecnología (programación, *hardware/software*, etc.). Progresivamente, se le proporcionaron al agente nuevos archivos, acompañados de un breve audio con instrucciones.

El objetivo era doble. Por un lado, se buscaba demostrar que el agente es capaz de analizar el nuevo contenido, comprenderlo en el contexto de la bóveda existente y determinar correctamente con qué notas preexistentes debía vincularlo. Por otro lado, y de igual importancia, se quería verificar que el agente no siempre establece conexiones. Si un nuevo artículo trata sobre un tema completamente ajeno a lo que ya existe en la bóveda, el resultado deseable es que se cree una nota aislada, sin vínculos forzados o incorrectos. Este comportamiento es crucial para mantener la integridad y la calidad del grafo de conocimiento.

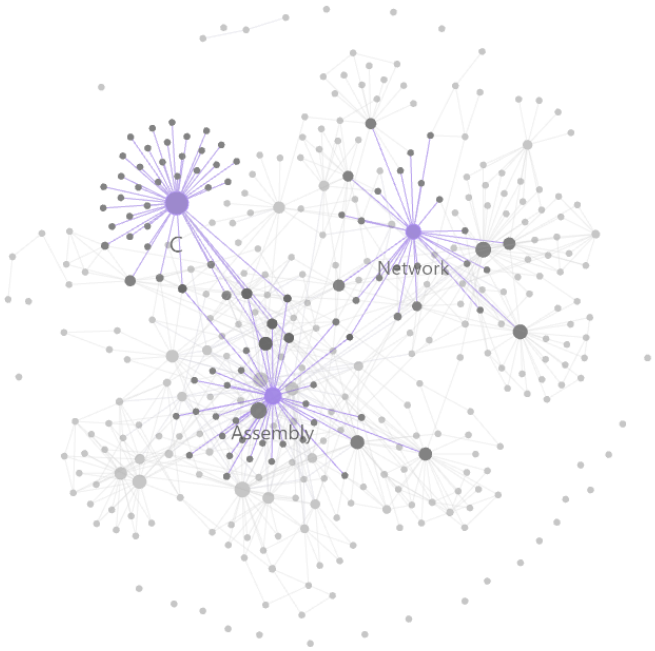


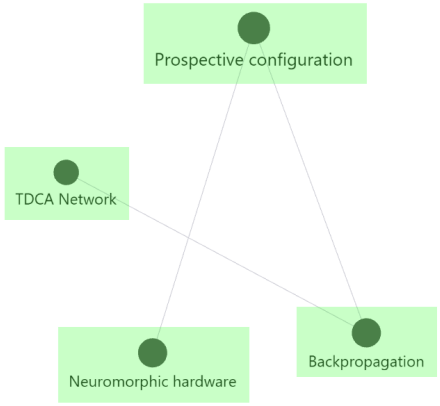
Figura 3.2. *Wiki* personal sobre temas de tecnología.

Para evaluar estos comportamientos, se realizaron dos secuencias de inserción. En la primera, se introdujeron artículos sobre *hardware* neuromórfico y alternativas a *backpropagation*, temas sin representación previa en la bóveda. Como se esperaba, el agente creó las notas correspondientes (visibles en su totalidad en la Figura 3.3A) pero no las vinculó con el grafo existente. En la segunda secuencia, se añadió una serie de artículos sobre los orígenes de las *Interaction Nets* y sus aplicaciones actuales. En este caso, el agente creó las siguientes notas, vinculándolas a 7 notas preexistentes:

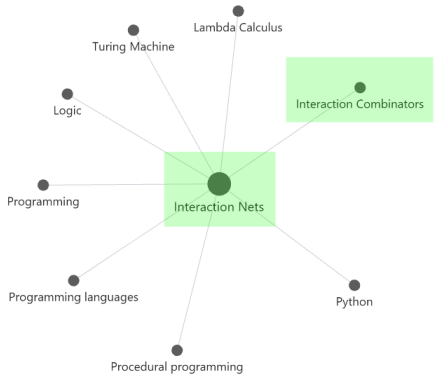
- *Interaction Nets.md*
- *Interaction Combinators.md*
- *HVM2.md*
- *Bend.md*

El resultado de esta integración se muestra parcialmente en la Figura 3.3B, donde se aprecian las nuevas notas conectadas al conocimiento previo.

En conjunto, estos resultados verifican la versatilidad del agente para determinar, de forma autónoma, qué notas existentes son candidatas idóneas para establecer vínculos con el nuevo contenido. Esta capacidad de discernimiento, reforzada por el optimizador de contexto, es fundamental para asegurar que la base de conocimiento crezca de manera coherente y relevante.



(A) Notas sobre *hardware* neuromórfico sin conexiones externas a las nuevas.



(B) Notas sobre *Interaction Nets* conectadas a notas existentes.

Figura 3.3. Comparación de resultados de inserción con las notas nuevas resaltadas.

Ambas vistas corresponden al grafo local de *Obsidian*. En el caso aislado (izquierda), se muestra con profundidad máxima (5), pero al no tener vínculos, solo aparecen los nodos resultantes de la secuencia de inserción. En el caso conectado (derecha), una profundidad de 1 es suficiente para visualizar las conexiones a notas existentes.

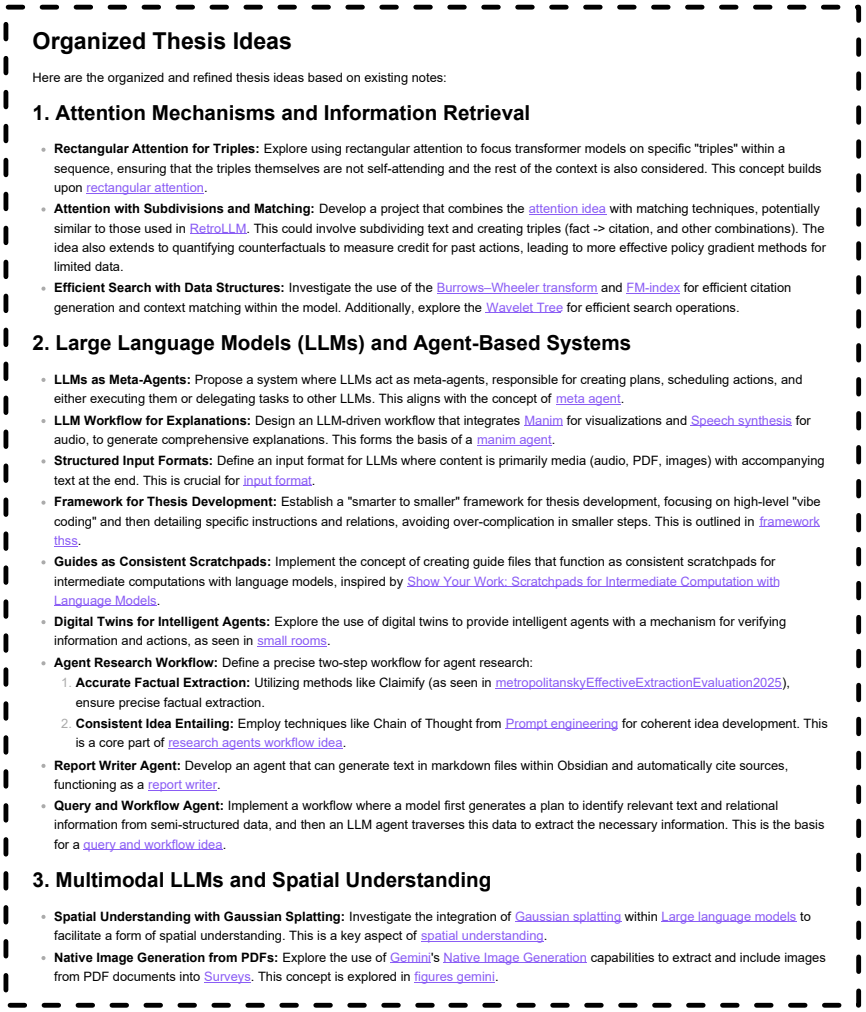
3.3. *Projects, Areas, Resources, Archived*

Este experimento refleja un caso de uso más cotidiano y de productividad personal, siguiendo la metodología *P.A.R.A. (Projects, Areas, Resources, Archived)* popularizada por Forte, *Building a Second Brain*. El objetivo era evaluar si el sistema puede ser utilizado para mantener una bóveda organizada bajo esta estructura, siguiendo instrucciones de voz. Para ello, se realizaron 4 peticiones donde cada una consistía en un archivo de audio, acompañado opcionalmente por archivos de contexto adicionales.

Se evaluó la capacidad del modelo para realizar diversas tareas, entre las que destacan:

- **Agregar detalles a proyectos existentes:** Por ejemplo, se solicitó añadir una nueva canción al proyecto de práctica de canto, proporcionando como contexto un documento con la letra adicional al audio con la instrucción.
- **Reescribir y sintetizar información:** Se instruyó por voz al modelo para que buscara información dispersa sobre un proyecto de tesis dentro de la bóveda. El contenido encontrado fue sintetizado en un nuevo documento. Además se propusieron nuevas ideas para complementar el trabajo en un documento adicional.
- **Crear nuevos proyectos:** Se crearon proyectos desde cero con indicaciones sobre sus objetivos y se vincularon a otros existentes. Por ejemplo, se generó un ’Nuevo proyecto de proyección vocal’ vinculado al proyecto de ’práctica de canto’.

Como resultado de estas 4 peticiones, se crearon un total de 6 notas nuevas, sin incluir los archivos de transcripción. La Figura 3.4 muestra un ejemplo de una de estas notas generadas, donde se sintetiza información dispersa y se vincula con otras notas.



Organized Thesis Ideas

Here are the organized and refined thesis ideas based on existing notes:

1. Attention Mechanisms and Information Retrieval

- **Rectangular Attention for Triples:** Explore using rectangular attention to focus transformer models on specific "triples" within a sequence, ensuring that the triples themselves are not self-attending and the rest of the context is also considered. This concept builds upon rectangular attention.
- **Attention with Subdivisions and Matching:** Develop a project that combines the attention idea with matching techniques, potentially similar to those used in RetroLLM. This could involve subdividing text and creating triples (fact -> citation, and other combinations). The idea also extends to quantifying counterfactuals to measure credit for past actions, leading to more effective policy gradient methods for limited data.
- **Efficient Search with Data Structures:** Investigate the use of the Burrows–Wheeler transform and FM-index for efficient citation generation and context matching within the model. Additionally, explore the Wavelet Tree for efficient search operations.

2. Large Language Models (LLMs) and Agent-Based Systems

- **LLMs as Meta-Agents:** Propose a system where LLMs act as meta-agents, responsible for creating plans, scheduling actions, and either executing them or delegating tasks to other LLMs. This aligns with the concept of meta agent.
- **LLM Workflow for Explanations:** Design an LLM-driven workflow that integrates Manim for visualizations and Speech synthesis for audio, to generate comprehensive explanations. This forms the basis of a manim agent.
- **Structured Input Formats:** Define an input format for LLMs where content is primarily media (audio, PDF, images) with accompanying text at the end. This is crucial for input format.
- **Framework for Thesis Development:** Establish a "smarter to smaller" framework for thesis development, focusing on high-level "vibe coding" and then detailing specific instructions and relations, avoiding over-complication in smaller steps. This is outlined in framework thss.
- **Guides as Consistent Scratchpads:** Implement the concept of creating guide files that function as consistent scratchpads for intermediate computations with language models, inspired by Show Your Work: Scratchpads for Intermediate Computation with Language Models.
- **Digital Twins for Intelligent Agents:** Explore the use of digital twins to provide intelligent agents with a mechanism for verifying information and actions, as seen in small rooms.
- **Agent Research Workflow:** Define a precise two-step workflow for agent research:
  1. **Accurate Factual Extraction:** Utilizing methods like Claimify (as seen in metropolitanskyEffectiveExtractionEvaluation2025), ensure precise factual extraction.
  2. **Consistent Idea Entailing:** Employ techniques like Chain of Thought from Prompt engineering for coherent idea development. This is a core part of research agents workflow idea.
- **Report Writer Agent:** Develop an agent that can generate text in markdown files within Obsidian and automatically cite sources, functioning as a report writer.
- **Query and Workflow Agent:** Implement a workflow where a model first generates a plan to identify relevant text and relational information from semi-structured data, and then an LLM agent traverses this data to extract the necessary information. This is the basis for a query and workflow idea.

3. Multimodal LLMs and Spatial Understanding

- **Spatial Understanding with Gaussian Splatting:** Investigate the integration of Gaussian splatting within Large language models to facilitate a form of spatial understanding. This is a key aspect of spatial understanding.
- **Native Image Generation from PDFs:** Explore the use of Gemini's Native Image Generation capabilities to extract and include images from PDF documents into Surveys. This concept is explored in figures gemini.

Figura 3.4. Ejemplo de una nota generada por el agente. Se sintetiza información sobre un proyecto de tesis y se enlaza a notas existentes.

Adicionalmente, este experimento contaba con instrucciones para la automatización de flujos. El agente no solo procesaba la petición, sino que también movía automáticamente el archivo de audio a una carpeta designada y generaba un archivo de transcripción en la misma ubicación, organizando así los ficheros procesados y probando sus capacidades para automatizar tareas. Incluir los archivos generados en este documento sería verboso y no aportaría más información que la confirmación de que fueron creados correctamente. Por ello, los resultados y los nuevos ficheros se pueden consultar en el repositorio del proyecto.

3.4. *Note augmented LLMs are computationally universal*

Este último experimento, de naturaleza más teórica, se inspira en la idea de que los modelos de lenguaje, cuando se aumentan con una memoria externa, pueden volverse computacionalmente universales (Schuurmans, *Memory Augmented Large Language Models Are Computationally Universal*).

Para explorar esta idea, se le pidió al agente que evaluara la Conjetura de Collatz para un número dado, utilizando una nota de *Obsidian* como su *memoria de trabajo* o registro. En cada paso de la secuencia, el agente leía el estado actual de la nota, calculaba el siguiente número, y sobrescribía la nota con el nuevo valor.

```
// variables
c = 0
k = 10

// instructions
if k is 1, you are going to finish the session

if k is even, you are going to replace k with k/2 and add 1 to c
otherwise, if k is odd, you are going to replace k with (3*k+1), and add 1 to c

once finished, you are going to open this file again and follow its instructions again (you must
always open the file, in case the file is modified)
```

Figura 3.5. Contenido inicial de la nota de trabajo para $n = 10$ en la evaluación de Collatz.

Se evaluó el caso para $n = 10$, que se resolvió correctamente en 6 iteraciones (Figuras 3.5 y 3.6). También se evaluaron casos más largos como $n = 11$ (14 iteraciones) y $n = 27$ (111 iteraciones). Este último caso excedió los límites de cuota de la *API* y requirió reiniciar el proceso, lo cual demostró una capacidad emergente del sistema: la posibilidad de dejar una tarea y seguirla luego, ya que el estado se preserva en la nota de *Obsidian*.

La intuición subyacente es que la capacidad de releer el estado desde un archivo fiable reduce drásticamente la probabilidad de error en un proceso iterativo largo, de manera análoga a cómo una *CPU* depende de leer y escribir correctamente en sus registros. En este caso se usó un modelo potente en razonamiento algebraico (*Gemini 2.5 Flash*). Si bien un modelo más modesto podría fallar en esta tarea matemática, la misma arquitectura podría permitirle automatizar con alta fiabilidad otra tarea iterativa que se alinee con sus fortalezas (por ejemplo, un análisis de sentimiento refinado a lo largo de múltiples revisiones), siempre que sea proficiente en las operaciones básicas de leer y escribir en su memoria externa.

```
 // variables
-c = 0
-k = 10
+c = 1
+k = 5
(...)
```

```
 // variables
-c = 1
-k = 5
+c = 2
+k = 16
(...)
```

```
 // variables
-c = 2
-k = 16
+c = 3
+k = 8
(...)
```

```
 // variables
-c = 3
-k = 8
+c = 4
+k = 4
(...)
```

```
 // variables
-c = 4
-k = 4
+c = 5
+k = 2
(...)
```

```
 // variables
-c = 5
-k = 2
+c = 6
+k = 1
(...)
```

Figura 3.6. Diferencia en el contenido de la nota entre los pasos de la evaluación para $n = 10$.

Conclusiones

El presente trabajo de tesis se propuso abordar el desafío de optimizar la integración de conocimiento en bases personales semiestructuradas, un proceso tradicionalmente manual y laborioso. La investigación partió de la pregunta de si un agente personalizable basado en Grandes Modelos de Lenguaje (*LLM*) podría automatizar y enriquecer este proceso de manera eficaz y adaptable a diversos paradigmas de toma de notas. La respuesta, materializada en el prototipo **lImporter**, es afirmativa y abre la puerta a una nueva generación de herramientas para la Gestión del Conocimiento Personal (*PKM*).

Se cumplieron los objetivos específicos planteados al inicio de la investigación. Primero, se desarrolló un *framework* flexible en forma del *plugin* 'lImporter' para *Obsidian.md*, capaz de procesar diversas fuentes de información. Segundo, se diseñó e implementó un agente autónomo basado en el paradigma *ReAct*, cuya arquitectura combina la estrategia *Plan-and-Solve* con la interacción fiable mediante *function calling*, y un sistema de optimización de contexto para consultar eficientemente la bóveda del usuario. Finalmente, la evaluación, realizada a través de una serie de experimentos cualitativos, demostró la versatilidad y eficacia del sistema para cumplir con los requisitos de distintos flujos de trabajo.

Los experimentos revelaron capacidades clave y matices importantes del sistema. La creación de una base de conocimiento sobre *Harry Potter* demostró la habilidad del agente para construir grafos de conocimiento estructurados y visualmente funcionales a partir de texto no estructurado. El mantenimiento de una *wiki* tecnológica evidenció una capacidad crucial: el discernimiento para crear conexiones relevantes cuando existen, y para mantener aislados los conceptos nuevos y no relacionados, preservando así la integridad del conocimiento. El experimento con la metodología *P.A.R.A.* validó su utilidad en escenarios de productividad personal más dinámicos y cotidianos. Por último, el experimento teórico de la Conjetura de *Collatz* desveló una capacidad emergente y profunda: la posibilidad de utilizar la base de notas como una memoria externa fiable, permitiendo al *LLM* realizar tareas iterativas y con estado de una manera robusta, análoga a un ciclo de computación.

A pesar de los resultados positivos, el trabajo presenta limitaciones inherentes. La calidad de las acciones del agente está intrínsecamente ligada a la capacidad de razonamiento del *LLM* subyacente. Asimismo, la dependencia de *APIs* externas implica costos monetarios y límites de uso que pueden restringir la ejecución de tareas muy extensas. La versatilidad del sistema, si bien es una fortaleza, también introduce una dependencia de la habilidad del usuario para formular instrucciones claras y efectivas (*prompting*). Finalmente, la evaluación fue de naturaleza cualitativa, centrándose en demostrar la viabilidad y el potencial, en lugar de realizar una comparativa cuantitativa rigurosa.

Las futuras líneas de investigación pueden expandir este trabajo en varias direcciones. La integración de modelos de lenguaje de código abierto y de ejecución local podría mitigar los costos y las preocupaciones de privacidad. Sería valioso dotar al agente de la capacidad de interactuar con otras herramientas y *plugins* del ecosistema de *Obsidian*, como *Dataview*, para realizar consultas estructuradas sobre la base de conocimiento. Se podría explorar el desarrollo de agentes proactivos que operen en segundo plano, sugiriendo conexiones o reorganizaciones sin una petición explícita del usuario. La evolución hacia sistemas multi-agente, donde diferentes agentes especializados colaboran en tareas complejas, representa una frontera prometedora.

En conclusión, este trabajo demuestra que la sinergia entre los *LLM* y los sistemas de *PKM* va más allá de la simple asistencia conversacional. Al dotar a los modelos de lenguaje de un cuerpo de conocimiento externo y la capacidad de actuar sobre él, se transforman en verdaderos colaboradores cognitivos. **lImporter** se erige como un prototipo funcional que valida este paradigma, mostrando un camino viable hacia la construcción de un *segundo cerebro* que no solo almacena información, sino que activamente ayuda a estructurarla, conectarla y generar nuevas ideas a partir de ella.

Bibliografía

Ahrens, Sönke. *How to Take Smart Notes: One Simple Technique to Boost Writing, Learning and Thinking – for Students, Academics and Nonfiction Book Writers*. Feb. de 2017 (ver páginas 1, 5).

Forte, Tiago. *Building a Second Brain*. Profile, jun. de 2022 (ver páginas 1, 5, 25).

Fraga, Felipe et al. «On the Automatic Generation of Knowledge Connections:» en: *Proceedings of the 25th International Conference on Enterprise Information Systems*. Prague, Czech Republic: SCITEPRESS - Science and Technology Publications, 2023, págs. 43-54. ISBN: 978-989-758-648-4. DOI: 10.5220/0011781100003467 (ver página 2).

Grundspenkis, Janis. «Agent Based Approach for Organization and Personal Knowledge Modelling: Knowledge Management Perspective». En: *Journal of Intelligent Manufacturing* 18.4 (jul. de 2007), págs. 451-457. ISSN: 0956-5515, 1572-8145. DOI: 10.1007/s10845-007-0052-6 (ver página 4).

Jansen, Renée S., Daniel Lakens y Wijnand A. IJsselsteijn. «An Integrative Review of the Cognitive Costs and Benefits of Note-Taking». En: *Educational Research Review* 22 (nov. de 2017), págs. 223-233. ISSN: 1747938X. DOI: 10.1016/j.edurev.2017.10.001 (ver página 4).

Pauk, Walter y Ross J. Q. Owens. *How to Study in College*. 10th Ed. Boston, MA: Cengage Learning, 2010. ISBN: 978-1-4390-8446-5 978-0-495-90302-4 (ver página 5).

Bush, Vannevar. *As We May Think*. Ene. de 1945 (ver página 5).

Davies, Stephen, Javier Velez-Morales y Roger King. «Building the Memex Sixty Years Later: Trends and Directions in Personal Knowledge Bases». En: (2005) (ver página 6).

Davies, Stephen. «Still Building the Memex». En: *Communications of the ACM* 54.2 (feb. de 2011), págs. 80-88. ISSN: 0001-0782, 1557-7317. DOI: 10.1145/1897816.1897840 (ver página 6).

Pyne, Yvette y Stuart Stewart. «Meta-Work: How We Research Is as Important as What We Research». En: *The British Journal of General Practice* 72.716 (feb. de 2022), págs. 130-131. ISSN: 0960-1643. DOI: 10.3399/bjgp22X718757 (ver página 6).

OpenAI et al. *GPT-4 Technical Report*. Mar. de 2024. DOI: 10.48550/arXiv.2303.08774. arXiv: 2303.08774 [cs] (ver página 7).

Team, Gemini et al. *Gemini: A Family of Highly Capable Multimodal Models*. Jun. de 2024. DOI: 10.48550/arXiv.2312.11805. arXiv: 2312.11805 [cs] (ver páginas 7, 14).

DeepSeek-AI et al. *DeepSeek-V3 Technical Report*. Dic. de 2024. DOI: 10.48550/arXiv.2412.19437. arXiv: 2412.19437 [cs] (ver página 7).

Bai, Jinze et al. *Qwen Technical Report*. Sep. de 2023. DOI: 10.48550/arXiv.2309.16609. arXiv: 2309.16609 [cs] (ver página 7).

Grattafiori, Aaron et al. *The Llama 3 Herd of Models*. Nov. de 2024. DOI: 10.48550/arXiv.2407.21783. arXiv: 2407.21783 [cs] (ver página 7).

Brown, Tom B. et al. «Language Models Are Few-Shot Learners». En: *Proceedings of the 34th International Conference on Neural Information Processing Systems*. NIPS '20. Red Hook, NY, USA: Curran Associates Inc., dic. de 2020, págs. 1877-1901. ISBN: 978-1-7138-2954-6 (ver página 7).

Min, Sewon et al. «Rethinking the Role of Demonstrations: What Makes In-Context Learning Work?» En: *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*. Ed. por Yoav Goldberg, Zornitsa Kozareva y Yue Zhang. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, dic. de 2022, págs. 11048-11064. DOI: 10.18653/v1/2022.emnlp-main.759 (ver página 7).

Kong, Aobo et al. *Better Zero-Shot Reasoning with Role-Play Prompting*. Mar. de 2024. DOI: 10.48550/arXiv.2308.07702. arXiv: 2308.07702 [cs] (ver página 7).

Lu, Albert et al. *Bounding the Capabilities of Large Language Models in Open Text Generation with Prompt Constraints*. Feb. de 2023. DOI: 10.48550/arXiv.2302.09185. arXiv: 2302.09185 [cs] (ver página 7).

Nye, Maxwell et al. *Show Your Work: Scratchpads for Intermediate Computation with Language Models*. Nov. de 2021. DOI: 10.48550/arXiv.2112.00114. arXiv: 2112.00114 [cs] (ver página 7).

Wei, Jason et al. *Chain-of-Thought Prompting Elicits Reasoning in Large Language Models*. Ene. de 2023. DOI: 10.48550/arXiv.2201.11903. arXiv: 2201.11903 [cs] (ver página 7).

Kojima, Takeshi et al. *Large Language Models Are Zero-Shot Reasoners*. Ene. de 2023. DOI: 10.48550/arXiv.2205.11916. arXiv: 2205.11916 [cs] (ver página 7).

Wang, Lei et al. *Plan-and-Solve Prompting: Improving Zero-Shot Chain-of-Thought Reasoning by Large Language Models*. Mayo de 2023. DOI: 10.48550/arXiv.2305.04091. arXiv: 2305.04091 [cs] (ver páginas 7, 17).

Fu, Yao et al. *Complexity-Based Prompting for Multi-Step Reasoning*. Ene. de 2023. DOI: 10.48550/arXiv.2210.00720. arXiv: 2210.00720 [cs] (ver página 8).

Zhou, Denny et al. *Least-to-Most Prompting Enables Complex Reasoning in Large Language Models*. Abr. de 2023. DOI: 10.48550/arXiv.2205.10625. arXiv: 2205.10625 [cs] (ver página 8).

Wang, Xuezhi et al. *Self-Consistency Improves Chain of Thought Reasoning in Language Models*. Mar. de 2023. DOI: 10.48550/arXiv.2203.11171. arXiv: 2203.11171 [cs] (ver página 8).

Yao, Shunyu et al. *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. Dic. de 2023. DOI: 10.48550/arXiv.2305.10601. arXiv: 2305.10601 [cs] (ver página 8).

Besta, Maciej et al. «Graph of Thoughts: Solving Elaborate Problems with Large Language Models». En: *Proceedings of the AAAI Conference on Artificial Intelligence* 38.16 (mar. de 2024), págs. 17682-17690. ISSN: 2374-3468, 2159-5399. DOI: 10.1609/aaai.v38i16.29720. arXiv: 2308.09687 [cs] (ver página 8).

Press, Ofir et al. *Measuring and Narrowing the Compositionality Gap in Language Models*. Oct. de 2023. DOI: 10.48550/arXiv.2210.03350. arXiv: 2210.03350 [cs] (ver página 8).

Madaan, Aman et al. *Self-Refine: Iterative Refinement with Self-Feedback*. Mayo de 2023. DOI: 10.48550/arXiv.2303.17651. arXiv: 2303.17651 [cs] (ver página 8).

Zheng, Huaixiu Steven et al. *Take a Step Back: Evoking Reasoning via Abstraction in Large Language Models*. Mar. de 2024. DOI: 10.48550/arXiv.2310.06117. arXiv: 2310.06117 [cs] (ver página 8).

Gao, Peizhong et al. *Meta Reasoning for Large Language Models*. Jun. de 2024. DOI: 10.48550/arXiv.2406.11698. arXiv: 2406.11698 [cs] (ver página 8).

Aytes, Simon A., Jinheon Baek y Sung Ju Hwang. *Sketch-of-Thought: Efficient LLM Reasoning with Adaptive Cognitive-Inspired Sketching*. Mar. de 2025. DOI: 10.48550/arXiv.2503.05179. arXiv: 2503.05179 [cs] (ver página 8).

Zhang, Yifan et al. *Cumulative Reasoning with Large Language Models*. Mar. de 2025. DOI: 10.48550/arXiv.2308.04371. arXiv: 2308.04371 [cs] (ver página 8).

Liu, Nelson F et al. «Lost in the Middle: How Language Models Use Long Contexts». En: (2024) (ver página 8).

Lewis, Patrick et al. *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. Abr. de 2021. DOI: 10.48550/arXiv.2005.11401. arXiv: 2005.11401 [cs] (ver página 8).

Team, Gemini et al. *Gemini 1.5: Unlocking Multimodal Understanding across Millions of Tokens of Context*. Dic. de 2024. DOI: 10.48550/arXiv.2403.05530. arXiv: 2403.05530 [cs] (ver página 9).

Yang, An et al. *Qwen2.5-1M Technical Report*. Ene. de 2025. DOI: 10.48550/arXiv.2501.15383. arXiv: 2501.15383 [cs] (ver página 9).

Agarwal, Rishabh et al. *Many-Shot In-Context Learning*. Oct. de 2024. DOI: 10.48550/arXiv.2404.11018. arXiv: 2404.11018 [cs] (ver página 9).

Chen, Tong et al. «Dense X Retrieval: What Retrieval Granularity Should We Use?» En: *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*. Ed. por Yaser Al-Onaizan, Mohit Bansal y Yun-Nung Chen. Miami, Florida, USA: Association for Computational Linguistics, nov. de 2024, págs. 15159-15177. DOI: 10.18653/v1/2024.emnlp-main.845 (ver páginas 9, 19).

Shen, Weizhou et al. *QwenLong-CPRS: Towards ∞-LLMs with Dynamic Context Optimization*. Mayo de 2025. DOI: 10.48550/arXiv.2505.18092. arXiv: 2505.18092 [cs] (ver páginas 9, 19).

Zhuang, Yufan et al. *Self-Taught Agentic Long Context Understanding*. Feb. de 2025. DOI: 10.48550/arXiv.2502.15920. arXiv: 2502.15920 [cs] (ver página 9).

Sun, Zhiqing et al. *Recitation-Augmented Language Models*. Feb. de 2023. DOI: 10.48550/arXiv.2210.01296. arXiv: 2210.01296 [cs] (ver página 9).

Karpas, Ehud et al. *MRKL Systems: A Modular, Neuro-Symbolic Architecture That Combines Large Language Models, External Knowledge Sources and Discrete Reasoning*. Mayo de 2022. DOI: 10.48550/arXiv.2205.00445. arXiv: 2205.00445 [cs] (ver página 9).

Gao, Luyu et al. *PAL: Program-aided Language Models*. Ene. de 2023. DOI: 10.48550/arXiv.2211.10435. arXiv: 2211.10435 [cs] (ver página 9).

Gou, Zhibin et al. *ToRA: A Tool-Integrated Reasoning Agent for Mathematical Problem Solving*. Feb. de 2024. DOI: 10.48550/arXiv.2309.17452. arXiv: 2309.17452 [cs] (ver página 9).

Yao, Shunyu et al. *ReAct: Synergizing Reasoning and Acting in Language Models*. Mar. de 2023. DOI: 10.48550/arXiv.2210.03629. arXiv: 2210.03629 [cs] (ver páginas 9, 16).

Shinn, Noah et al. *Reflexion: Language Agents with Verbal Reinforcement Learning*. Mar. de 2023. DOI: 10.48550/arXiv.2303.11366. arXiv: 2303.11366 (ver página 9).

Sumers, Theodore R. et al. *Cognitive Architectures for Language Agents*. Mar. de 2024. DOI: 10.48550/arXiv.2309.02427. arXiv: 2309.02427 [cs] (ver página 10).

Van Durme, Benjamin y Lenhart Schubert. «Open Knowledge Extraction through Compositional Language Processing». En: *Semantics in Text Processing. STEP 2008 Conference Proceedings*. Ed. por Johan Bos y Rodolfo Delmonte. College Publications, 2008, págs. 239-254 (ver página 10).

Song, Linfeng et al. «OpenFact: Factuality Enhanced Open Knowledge Extraction». En: *Transactions of the Association for Computational Linguistics* 11 (jun. de 2023), págs. 686-702. ISSN: 2307-387X. DOI: 10.1162/tacl_a_00569 (ver página 10).

Qian, Kun et al. *Open Domain Knowledge Extraction for Knowledge Graphs*. Oct. de 2023. DOI: 10.48550/arXiv.2312.09424. arXiv: 2312.09424 [cs] (ver página 10).

Zhang, Bowen y Harold Soh. *Extract, Define, Canonicalize: An LLM-based Framework for Knowledge Graph Construction*. Oct. de 2024. DOI: 10.48550/arXiv.2404.03868. arXiv: 2404.03868 [cs] (ver página 10).

Kommineni, Vamsi Krishna, Birgitta König-Ries y Sheeba Samuel. *From Human Experts to Machines: An LLM Supported Approach to Ontology and Knowledge Graph Construction*. Mar. de 2024. DOI: 10.48550/arXiv.2403.08345. arXiv: 2403.08345 [cs] (ver página 11).

Li, Zixuan et al. *KnowCoder: Coding Structured Knowledge into LLMs for Universal Information Extraction*. Mar. de 2024. DOI: 10.48550/arXiv.2403.07969. arXiv: 2403.07969 [cs] (ver página 11).

Luo, Yujie et al. *OneKE: A Dockerized Schema-Guided LLM Agent-based Knowledge Extraction System*. Feb. de 2025. DOI: 10.48550/arXiv.2412.20005. arXiv: 2412.20005 [cs] (ver página 11).

Li, Diya et al. *Automated Clinical Data Extraction with Knowledge Conditioned LLMs*. Nov. de 2024. DOI: 10.48550/arXiv.2406.18027. arXiv: 2406.18027 [cs] (ver página 11).

Feng, Xiaohan, Xixin Wu y Helen Meng. *Ontology-Grounded Automatic Knowledge Graph Construction by LLM under Wikidata Schema*. Dic. de 2024. DOI: 10.48550/arXiv.2412.20942. arXiv: 2412.20942 [cs] (ver página 11).

McCusker, Jamie. *LOKE: Linked Open Knowledge Extraction for Automated Knowledge Graph Construction*. Nov. de 2023. DOI: 10.48550/arXiv.2311.09366. arXiv: 2311.09366 [cs] (ver página 11).

Abolhasani, Mohammad Sadeq y Rong Pan. *Leveraging LLM for Automated Ontology Extraction and Knowledge Graph Generation*. Dic. de 2024. DOI: 10.48550/arXiv.2412.00608. arXiv: 2412.00608 [cs] (ver página 11).

Shu, Dong et al. *Knowledge Graph Large Language Model (KG-LLM) for Link Prediction*. Ago. de 2024. DOI: 10.48550/arXiv.2403.07311. arXiv: 2403.07311 [cs] (ver página 11).

He, Zhongmou et al. *LinkGPT: Teaching Large Language Models To Predict Missing Links*. Jun. de 2024. DOI: 10.48550/arXiv.2406.04640. arXiv: 2406.04640 [cs] (ver página 11).

Bi, Baolong et al. *LPNL: Scalable Link Prediction with Large Language Models*. Feb. de 2024. DOI: 10.48550/arXiv.2401.13227. arXiv: 2401.13227 [cs] (ver página 11).

Carta, Salvatore et al. *Iterative Zero-Shot LLM Prompting for Knowledge Graph Construction*. Jul. de 2023. DOI: 10.48550/arXiv.2307.01128. arXiv: 2307.01128 [cs] (ver página 12).

Park, Gilchan et al. *Enhancing Future Link Prediction in Quantum Computing Semantic Networks through LLM-Initiated Node Features*. Oct. de 2024. DOI: 10.48550/arXiv.2410.04251. arXiv: 2410.04251 [cs] (ver página 12).

Zhu, Yuqi et al. «LLMs for Knowledge Graph Construction and Reasoning: Recent Capabilities and Future Opportunities». En: *World Wide Web* 27.5 (sep. de 2024), pág. 58. ISSN: 1386-145X, 1573-1413. DOI: 10.1007/s11280-024-01297-w (ver página 12).

Machado, Marcelo et al. «{LLM Store}: Leveraging Large Language Models as Sources of {Wikidata}-Structured Knowledge». En: (2024) (ver página 12).

Geng, Saibo et al. *Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning*. Ene. de 2024. DOI: 10.48550/arXiv.2305.13971. arXiv: 2305.13971 [cs] (ver página 17).

Schuurmans, Dale. *Memory Augmented Large Language Models Are Computationally Universal*. Ene. de 2023. DOI: 10.48550/arXiv.2301.04589. arXiv: 2301.04589 [cs] (ver página 27).