31 Aniversario 1995 - 2026
CIENCIA, INGENIERÍA Y DESARROLLO TEC LERDO
Año: 2026 Volumen: 2 Numero: 12 ISSN: 2448-623X Revista Digital Universitaria

Diseño e Implementación de KG-RAG Dinámico: Una Extensión del Paradigma RAG con Memoria Persistente mediante Grafo de Conocimiento Bitemporal

Design and Implementation of Dynamic KG-RAG: Extending the RAG Paradigm with Persistent Memory through a Bitemporal Knowledge Graph

Caldera-Sánchez, C. J.[1]

Datos de Adscripción:
[1] ✉Caldera Sánchez, Christian Javier
Narvalla Innovación S.A.P.I. de C.V. Gómez Palacio, Durango. Mexico.
christian.caldera@outlook.com
ORCID: https://orcid.org/0009-0000-2638-9737

Resumen **- Los sistemas de Generación Aumentada por Recuperación (RAG) permiten consultar fuentes externas para generar respuestas fundamentadas; sin embargo, carecen de mecanismos para actualizar el conocimiento entre interacciones, lo que limita su personalización y continuidad a largo plazo. Este artículo presenta el diseño e implementación de un motor de memoria persistente multicapa, denominado KG-RAG Dinámico, que extiende este paradigma mediante la incorporación de memoria dinámica basada en un grafo de conocimiento bitemporal. El objetivo es desarrollar una arquitectura capaz de retener, estructurar y recuperar conocimiento a través de sesiones independientes. Para ello, se siguió un enfoque en cuatro fases: revisión del estado del arte, definición arquitectónica, implementación y validación experimental. La propuesta organiza la memoria en tres capas: transcripciones brutas inmutables, un grafo bitemporal que modela entidades, relaciones y su evolución en el tiempo, y una capa de recuperación híbrida que integra búsqueda semántica mediante embeddings, búsqueda léxica y recorrido de grafo. El contexto recuperado se inyecta sistemáticamente antes de la generación de respuestas. El grafo se construye de forma automática y asíncrona mediante un pipeline de extracción de entidades y relaciones, eliminando la necesidad de indexación manual. Los resultados muestran una recuperación consistente de contexto entre sesiones y una representación coherente de hechos cambiantes en el tiempo. La viabilidad se valida mediante un prototipo implementado como agente conversacional. Se concluye que KG-RAG Dinámico amplía eficazmente el paradigma RAG y es aplicable a sistemas que requieren memoria estructurada, persistente y evolutiva.**

Palabras Clave **- Grafo de Conocimiento Bi-temporal, KG-RAG Dinámico, Memoria Persistente Dinámica, Modelos de Lenguaje, Recuperación Híbrida, Sistemas de Inteligencia Artificial.**

Abstract **- Retrieval-Augmented Generation (RAG) systems allow querying external sources to generate grounded responses; however, they lack mechanisms to update knowledge between interactions, limiting long-term personalization and continuity. This article presents the design and implementation of a multi-layer persistent memory engine called Dynamic KG-RAG, which extends this paradigm by incorporating dynamic memory based on a bitemporal knowledge graph. The objective is to develop an architecture capable of retaining, structuring, and retrieving knowledge across independent sessions. To this end, a four-phase approach was followed: state-of-the-art review, architectural definition, implementation, and experimental validation. The proposal organizes memory into three layers: immutable raw transcripts, a bitemporal graph that models entities, relationships, and their evolution over time, and a hybrid retrieval layer that integrates semantic search through embeddings, lexical search, and graph traversal. Retrieved context is systematically injected before response generation. The graph is built automatically and asynchronously through an entity and relationship extraction pipeline, eliminating the need for manual indexing. Results show consistent context retrieval across sessions and coherent representation of facts changing over time. Viability is validated through a prototype implemented as a conversational agent. It is concluded that Dynamic KG-RAG effectively extends the RAG paradigm and is applicable to systems requiring structured, persistent, and evolving memory.**

Keywords **- Bitemporal Knowledge Graph, Dynamic KG-RAG, Dynamic Persistent Memory, Hybrid Retrieval, Intelligent Systems, Language Models.**

I. INTRODUCCIÓN

Los modelos de lenguaje de gran tamaño (Large Language Model, LLM) permiten a los sistemas de software generar interfaces de usuario basadas en lenguaje natural. Su uso se ha extendido a agentes de inteligencia artificial conversacionales que permiten ejecutar lógica y herramientas de forma no lineal. Estos agentes son de gran utilidad para interactuar con sistemas externos y automatizar procesos.

Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

Sin embargo, estos sistemas cuentan con grandes desventajas conocidas, como lo son: alucinaciones (inventan información falsa), falta de información actualizada, privacidad de datos, mínima o nula retención de información entre sesiones de trabajo, entre otras. Para estos cuatro problemas en particular, se han propuesto diferentes alternativas de contención, una de ellas son los sistemas de Recuperación Aumentada por Generación (Retrieval-Augmented Generation, RAG). Los sistemas RAG permiten el procesamiento y almacenamiento de fragmentos de texto en forma de vectores semánticos (embeddings) para su futura recuperación y generación de respuestas mediante el uso de LLMs, permitiendo así la reducción de alucinaciones, actualización y extensión de información almacenada, además de mantener la información dentro de la infraestructura de la organización.

Sin duda los sistemas RAG son de gran ayuda para mitigar algunas de las desventajas de los LLMs; sin embargo, este tipo de sistemas también tiene límites. Al depender exclusivamente de búsqueda por similitud semántica, su efectividad disminuye a medida que crece la cantidad de información almacenada: entre más documentos hay, más probable es que el sistema recupere fragmentos irrelevantes en lugar de los correctos (Reimers y Gurevych, 2021). Además, estos sistemas dependen fuertemente de la calidad de los datos ingresados y presentan falta de jerarquías y límites de las ventanas de contexto.

Aunque los sistemas RAG cumplen bien su función de procesar documentos para su futura recuperación, se ven limitados en escenarios donde la conversación activa y la memoria de hechos concretos toman relevancia central. Allí, la recuperación por similitud semántica resulta insuficiente, y se requiere una arquitectura de memoria distinta: una capaz de almacenar, actualizar y razonar sobre hechos específicos a lo largo del tiempo.

Trabajos recientes han comenzado a abordar esta brecha. MemGPT (Packer et al., 2023) propone una jerarquía de memoria inspirada en sistemas operativos, pero gestiona el almacenamiento como texto plano y fragmentos vectorizados, sin un modelo explícito de entidades y relaciones. Microsoft GraphRAG (Edge et al., 2024) introduce grafos de conocimiento para RAG (KG-RAG), pero construye índices estáticos offline sin capacidad de evolución dinámica. Mem0 (Chhikara et al., 2025) ofrece memoria escalable para agentes en producción, pero sin un modelo temporal explícito que distinga cuándo un hecho fue verdadero de cuándo dejó de serlo. Surveys recientes sobre mecanismos de memoria en agentes LLM (Zhang et al., 2025) organizan el área en torno a operaciones de lectura, escritura y manejo de información, sin tratar el modelo bitemporal como categoría de análisis.

La integración de bitemporalidad, es decir, el seguimiento simultáneo de cuándo un hecho es válido en el mundo real (valid time) y cuándo fue registrado o invalidado en el sistema (transaction time), permanece como un área poco explorada en sistemas conversacionales.

El sistema más cercano a la propuesta presentada en este artículo es Graphiti/Zep (Rasmussen et al., 2025), que emplea un grafo temporal para memoria de agentes con un modelo bitemporal. Sin embargo, difiere en decisiones arquitectónicas fundamentales: requiere un motor de grafos externo (Neo4j por defecto) junto con almacenamiento vectorial separado, lo que introduce complejidad operativa y riesgos de consistencia transaccional; extrae entidades y relaciones sin una descomposición explícita en subtareas especializadas; y depende de que el agente invoque herramientas para acceder a la memoria, en lugar de inyectar el contexto de forma obligatoria antes de la generación. Estas diferencias y sus implicaciones se analizan en detalle en la sección de resultados y discusión de este documento.

Este artículo presenta el diseño e implementación de KG-RAG Dinámico, un motor de memoria persistente multicapa que extiende el paradigma RAG mediante tres contribuciones principales: Un grafo de conocimiento bitemporal que modela la evolución del conocimiento a través del tiempo, un pipeline de extracción automática y asíncrona que construye el grafo sin intervención manual mediante la descomposición en subtareas especializadas, y un mecanismo de recuperación híbrida que fusiona búsqueda semántica, léxica y por grafo con inyección obligatoria de contexto previo a la generación. La viabilidad del sistema se valida mediante un prototipo funcional implementado como agente conversacional, evaluado con un framework de 35 pruebas automatizadas que cubren extracción, recuperación, evolución temporal y normalización de predicados. El código fuente se libera como repositorio público para facilitar la reproducibilidad.

II. PARTE TÉCNICA DEL ARTÍCULO

Se presenta a continuación el diseño de arquitectura utilizado para la implementación del sistema.

2.1 Visión General y Stack Tecnológico

KG-RAG Dinámico se implementa como un motor de memoria persistente multicapa orquestado por un agente conversacional denominado Mnemo. El sistema recibe mensajes del usuario, los procesa para extraer conocimiento estructurado de forma asíncrona, y recupera contexto relevante de conversaciones anteriores para inyectarlo antes de cada respuesta. Conceptualmente, el sistema se organiza en tres capas: transcripciones inmutables que conservan el historial completo de mensajes, un grafo de conocimiento bitemporal que estructura entidades y hechos con su evolución temporal, y una capa de recuperación híbrida que fusiona señales semánticas, léxicas y de grafo antes de la generación de respuestas. El prototipo se desarrolló sobre un stack de tecnologías maduras y con gran aceptación en la comunidad de desarrollo de software, priorizando componentes de código abierto y ejecución local donde fuera posible. La Tabla 1 detalla el stack tecnológico completo, mientras que la Figura 1 muestra las interacciones entre las capas y componentes descritos. En las siguientes subsecciones se profundizará en cada uno de los flujos principales del sistema. Adicionalmente, la Figura 2 presenta la interfaz de usuario del agente conversacional del sistema en uso.

Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

Tabla 1
Tecnologías utilizadas dentro del sistema.

Capa	Tecnología	Rol
Backend API	ASP.NET v9	REST API con streaming SSE
Agente	Anthropic SDK	Tool-use loop con Claude
Base de datos	PostgreSQL v17 + pgvector	Almacenamiento unificado + búsqueda vectorial
Embeddings	Ollama (snowflake-arctic-embed2)	Vectores de 1024 dimensiones, MTEB ~55
NER local	GLiNER (gliner-bi-large-v2.0)	Extracción de entidades
LLM	Claude Sonnet + Opus	Razonamiento, extracción, clasificación semántica
Frontend	Angular v21	SPA con visualización de grafo via Cytoscape.js
Infraestructura	Docker Compose	Orquestación de infraestructura de servicios

Figura 1
Diagrama de componentes del sistema KG-RAG Dinámico.

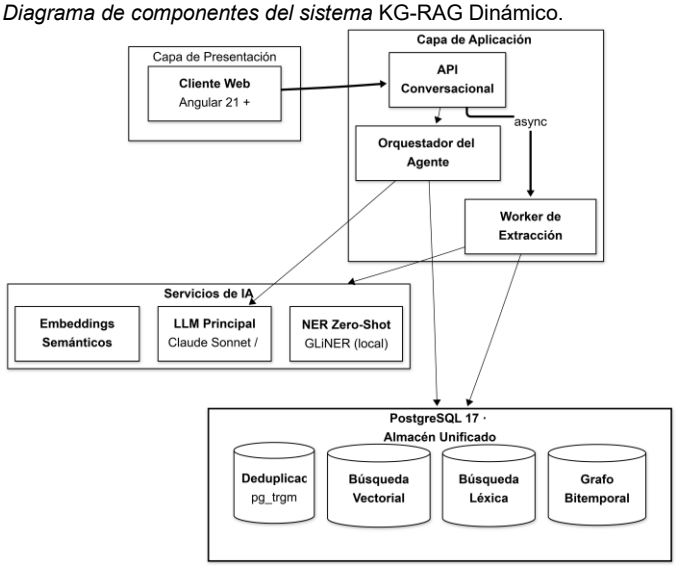
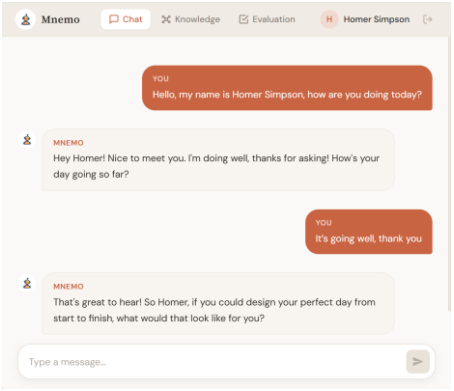


Figura 2
Agente conversacional Mnemo en uso.



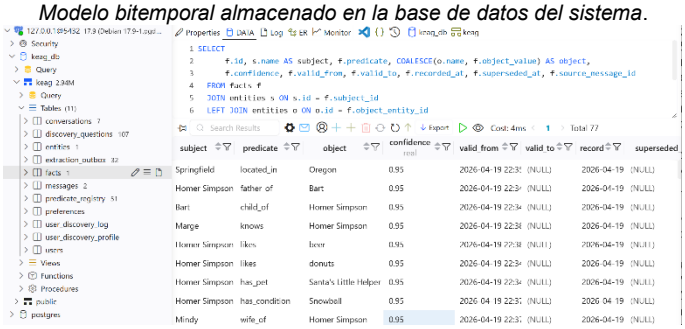
2.2 Modelo de datos bitemporal

El grafo de conocimiento se estructura como tripletas (sujeto, predicado, objeto) donde cada hecho porta cuatro marcas temporales: *valid_from* y *valid_to* representan la vigencia del hecho en el mundo real, *recorded_at* captura el momento de extracción por el sistema, y *superseded_at* registra cuándo un hecho posterior lo reemplazó (*NULL* indica hecho vigente). Este modelo se fundamenta en la distinción entre valid time y transaction time formalizada por Snodgrass y Ahn (1985), e incorporada posteriormente al estándar SQL:2011 como soporte nativo para tablas con periodos de validez y versionado del sistema (Kulkarni y Michels, 2012). En el contexto específico de grafos de conocimiento, Leblay y Chekol (2018) demostraron la viabilidad de este modelo bitemporal sobre 6.7 millones de hechos temporales de Wikidata.

La forma en que un hecho evoluciona depende de si el predicado acepta un solo valor o varios, según lo define el registro de predicados canónicos *predicate_registry*. Para predicados que solo aceptan un valor a la vez *lives_in*, *works_at*, *has_age*, un hecho nuevo reemplaza al anterior: el existente se marca con *superseded_at = now()* y el nuevo se crea con *superseded_at = NULL*. Ambos coexisten en la base de datos sin eliminarse. Para predicados que aceptan múltiples valores simultáneos *likes*, *has_hobby*, *parent_of*, los valores nuevos se acumulan junto con los anteriores.

Como ejemplo: el usuario menciona "Homero tiene 39 años" en enero 2025. Se crea *fact(has_age, 39, recorded_at=2025-01, superseded_at=NULL)*. Meses después dice "Homero cumplió 40". El hecho anterior se marca como supersedido y se crea uno nuevo vigente. Ambos persisten, lo que permite reconstruir el estado del grafo en cualquier punto temporal. La consulta de hechos vigentes es directa *WHERE superseded_at IS NULL*, mientras que consultas históricas combinan los cuatro campos temporales para responder preguntas como "¿dónde vivía el usuario en 2024?". En la Figura 3, se muestra una consulta con los hechos que contiene varios de los campos descritos en esta sección.

Figura 3
Modelo bitemporal almacenado en la base de datos del sistema.



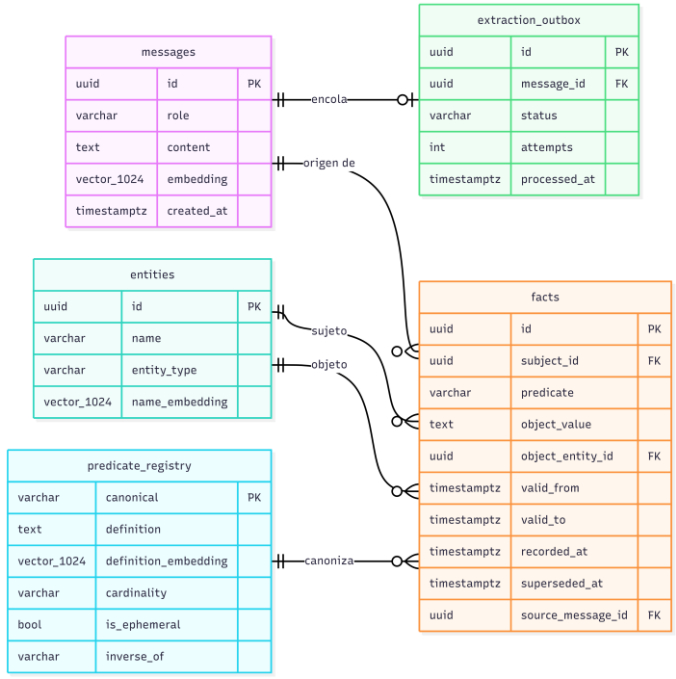
La Figura 4 presenta el modelo entidad-relación del núcleo del grafo de conocimiento. El esquema evidencia tres decisiones arquitectónicas centrales del sistema: la tabla *facts* incorpora las cuatro marcas temporales directamente como columnas, el registro *predicate_registry* almacena las definiciones canónicas

Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

junto con sus embeddings vectoriales, y la cola *extraction_outbox* se vincula a *messages* mediante clave foránea para garantizar persistencia atómica dentro de la misma transacción.

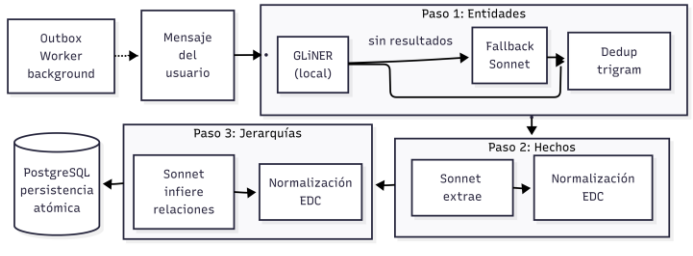
Figura 4
Modelo entidad-relación del núcleo del grafo de conocimiento.



2.3 Pipeline de extracción

La construcción del grafo se realiza mediante un pipeline asíncrono de tres pasos que transforma texto conversacional en conocimiento estructurado (Figura 5).

Figura 5
Pipeline de extracción y almacenamiento de datos.



En el primer paso se identifican las entidades mencionadas en el mensaje (personas, lugares, organizaciones, etc.). Para esto se utiliza GLiNER (Zaratiana et al., 2024), un modelo de reconocimiento de entidades que corre localmente. Si no detecta resultados, Claude Sonnet (Anthropic, 2024) actúa como respaldo. Se detectan aliases (apodos, nombres de soltera) y se verifican contra la base de datos para evitar duplicados. La entidad del usuario se incluye siempre, incluso si no se menciona explícitamente, para que el siguiente paso pueda resolver correctamente pronombres como "yo", "mi" o "me".

En el segundo paso, Claude Sonnet recibe el mensaje junto con las entidades detectadas y extrae relaciones en forma de tripletas (sujeto, predicado, objeto). Cada predicado extraído pasa por un proceso de normalización llamado EDC (Extract-Define-Canonicalize): un LLM genera una definición semántica del predicado, esa definición se vectoriza y se compara contra un registro de 56 predicados canónicos. Si coincide con uno existente se registra como sinónimo; si no, se crea uno nuevo. Predicados efímeros como "dijo" o "mencionó" se descartan automáticamente.

En el tercer paso, Claude Sonnet infiere relaciones implícitas por conocimiento del mundo (por ejemplo, *Springfield → located_in → Oregon*). Los hechos inferidos pasan por el mismo proceso de normalización del paso anterior.

El pipeline se ejecuta en background mediante el patrón outbox transaccional (Richardson, 2018): el mensaje y la entrada en la cola de extracción se persisten en la misma transacción de base de datos. Un worker-role procesa la cola cada 5 segundos con máximo 3 reintentos. Este diseño desacopla la latencia de extracción de la experiencia conversacional: el usuario recibe su respuesta sin esperar a que el grafo se actualice. La descomposición en subtareas especializadas sigue los principios de Khot et al. (2023), y resolvió errores frecuentes de resolución de pronombres y clasificación que se observaban con un prompt unificado.

2.4 Recuperación híbrida pre-fetch

El mecanismo de recuperación combina tres tipos de búsqueda que se ejecutan en paralelo ante cada mensaje del usuario: búsqueda semántica (similitud del coseno sobre embeddings de mensajes históricos), búsqueda léxica (Full-Text Search nativo de PostgreSQL), y búsqueda en el grafo de conocimiento (coincidencia por similaridad sobre nombres de entidades, seguida de la recuperación de sus hechos vigentes). Los resultados de las tres búsquedas se fusionan mediante Reciprocal Rank Fusion (Cormack et al., 2009), un método que combina rankings de distintas fuentes asignando puntuación inversamente proporcional a la posición de cada resultado. Se aplica además un decaimiento temporal exponencial que prioriza recuerdos recientes sobre los antiguos. Los 10 fragmentos con mayor puntuación se ensamblan en un bloque XML estructurado que se inyecta en el prompt del agente antes de generar la respuesta.

Esta inyección es obligatoria: el contexto de memoria se entrega al modelo antes de la generación, sin depender de que el agente decida buscarlo por cuenta propia. Esta decisión se fundamenta en evidencia del Berkeley Function Calling Leaderboard (Patil et al., 2025), que documenta que múltiples modelos omiten invocaciones de herramientas en escenarios multi-turno, y en T-Eval (Chen et al., 2024), que identifica cuellos de botella específicos en la selección y parametrización de herramientas por parte de LLMs. La inyección pre-fetch garantiza un piso mínimo de contexto relevante independiente del comportamiento del modelo. Adicionalmente, el agente dispone de 12 herramientas para profundizar bajo demanda en hasta 5 rondas de interacción.

Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

La combinación de las tres búsquedas complementarias se justifica por los hallazgos de Luan et al. (2021), quienes demuestran que las representaciones semánticas y léxicas capturan señales de relevancia distintas, y por el benchmark BEIR (Thakur et al., 2021), que confirmó que ningún método individual de búsqueda domina en todos los escenarios.

2.5 Framework de evaluación

Para validar las capacidades del sistema de forma controlada y reproducible, se diseñó un framework de evaluación integrado directamente en la aplicación. El framework ejecuta 35 pruebas automatizadas organizadas en cuatro categorías: extracción de entidades y hechos (15 pruebas), recuperación de información entre sesiones (8), evolución temporal de hechos (8) y normalización de predicados (4).

Las pruebas se ejecutan contra un perfil ficticio detallado (Homer Simpson) que permite verificar escenarios complejos de conocimiento en condiciones reproducibles: relaciones familiares, jerarquías laborales, preferencias, aliases y cambios de atributos a lo largo del tiempo. El framework invoca el pipeline de extracción directamente, sin pasar por el outbox worker, para eliminar condiciones de carrera y retardos no deterministas.

La verificación de resultados emplea tres estrategias de matching diseñadas para tolerar la variabilidad de la extracción basada en LLM: matching fuzzy de predicados mediante un mapa de equivalencias que define 21 conjuntos cubriendo aproximadamente 80 predicados (por ejemplo, "*likes*", "*enjoys*", "*is_fond_of*" y "*adores*" se tratan como equivalentes), matching parcial case-insensitive para nombres de entidades, y verificación directa de estados de supersesión temporal (*superseded_at IS NULL* para hechos vigentes, *superseded_at IS NOT NULL* para hechos reemplazados).

Cada prueba recibe uno de cinco estados posibles: aprobada, parcial (alguna verificación falla, pero la entidad se extrajo correctamente), fallida, omitida (una dependencia previa no pasó) o error (excepción durante la ejecución). Las pruebas se optimizaron mediante batching: múltiples hechos se consolidan en mensajes individuales cuando es semánticamente apropiado, reduciendo las invocaciones del pipeline de 42 a 31 (reducción del 26% en llamadas a la API). Los resultados cuantitativos de este framework se presentan en la sección III de resultados y discusión.

2.6 Parámetros de configuración

El comportamiento del sistema se controla mediante un conjunto de parámetros configurables que rigen los umbrales de calidad, los límites operativos y la lógica de recuperación. Estos valores se ajustaron durante el desarrollo del prototipo mediante experimentación iterativa y se documentan en la Tabla 2 para facilitar la reproducibilidad. Cada parámetro se expone en las clases de configuración del proyecto y puede modificarse sin recompilar el sistema.

Tabla 2

Parámetros clave del sistema y sus valores de configuración.

Parámetro	*Valor*	*Función*
GlinerThreshold	0.4	Puntaje mínimo para aceptar una entidad detectada por GLiNER
TrigramSimilarityThreshold	0.6	Similaridad requerida para deduplicar entidades existentes
DefinitionMatchThreshold	0.15	Distancia coseno máxima para considerar dos predicados equivalentes
BootstrapPredicates	56	Predicados canónicos pre-cargados (incluye 4 efímeros)
EmbeddingDimensions	1024	Dimensión de los vectores generados por snowflake-arctic-embed2
RrfK	60	Constante del algoritmo Reciprocal Rank Fusion
TemporalDecayLambda	0.003	Tasa de decaimiento temporal (~333 días para 50% de reducción)
HybridSearchTopK	10	Fragmentos de memoria inyectados como contexto por mensaje
RecentMessagesCount	12	Mensajes recientes incluidos en el contexto del agente
MaxToolRounds	5	Rondas máximas de uso de herramientas por respuesta
PollIntervalSeconds	5	Frecuencia de sondeo de la cola de extracción
MaxRetries	3	Reintentos antes de marcar una extracción como fallida
MaxFactsPerEntity	15	Límite de hechos recuperados por entidad en una consulta
MinLength	15	Longitud mínima en caracteres para encolar extracción

La selección de estos valores responde a un balance entre precisión y permisividad del sistema. Umbrales más estrictos (por ejemplo, reducir *DefinitionMatchThreshold* por debajo de 0.15) producen un registro de predicados más amplio con mayor duplicación semántica; umbrales más laxos colapsan predicados distintos en categorías excesivamente generales, un problema inherente a la canonicalización de predicados abiertos en grafos de conocimiento (Vashishth et al., 2018). El valor de 0.15 se estableció experimentalmente como el punto donde la tasa de falsos positivos en matching de predicados desciende a niveles aceptables sin sacrificar cobertura.

2.7 Distribución de modelos por tarea

El sistema emplea dos modelos de la familia Claude 3, Sonnet y Opus (Anthropic, 2024) distribuidos según la complejidad semántica de cada tarea, además de modelos locales para embeddings y reconocimiento de entidades. Esta distribución responde a un balance entre costo de inferencia y capacidad requerida: Sonnet cubre la mayoría de las tareas con precisión adecuada, mientras que Opus se reserva exclusivamente para la clasificación de predicados en el pipeline EDC, donde la experimentación interna demostró que Sonnet produce clasificaciones inconsistentes en distinciones semánticas finas (cardinalidad, tipos esperados, predicados inversos, detección de efímeros). La Tabla 3 detalla la asignación de modelos por tarea.

Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

Tabla 3
Distribución de modelos por tarea dentro del sistema.

Tarea	Modelo	Ubicación
Conversación principal (Mnemo)	Claude Sonnet	Agente conversacional
Extracción de hechos	Claude Sonnet	Pipeline de extracción, paso 2
Inferencia de jerarquías	Claude Sonnet	Pipeline de extracción, paso 3
Extracción de entidades (fallback)	Claude Sonnet	Pipeline de extracción, paso 1
Detección de aliases	Claude Sonnet	Pipeline de extracción, paso 1
Clasificación de predicados (EDC)	Claude Opus	Normalización de predicados
Generación de embeddings	snowflake-arctic-embed2 (local)	Embeddings
Reconocimiento de entidades (NER)	GLiNER (local)	Pipeline de extracción, paso 1

La restricción de Opus únicamente al paso de clasificación de predicados produce un impacto económico significativo: aunque Opus cuesta aproximadamente cinco veces más por token que Sonnet, su uso acotado a un solo paso del pipeline mantiene el costo total por ejecución del framework de evaluación entre 0.50 y 0.70 USD. Adicionalmente, el sistema emplea prompt caching de Anthropic en las tareas con system prompts estáticos, particularmente en la clasificación de predicados mediante Opus y en la extracción de hechos mediante Sonnet. Dado que el system prompt se reutiliza en cada invocación, el caching reduce aproximadamente 90% los tokens facturados en llamadas sucesivas, lo que mitiga el costo de usar Opus en un pipeline que se invoca miles de veces durante la operación normal del sistema. Los embeddings y la detección de entidades se ejecutan localmente sobre GPU, sin costo por token ni dependencia de APIs externas, lo que reduce la superficie de dependencia comercial a un solo proveedor (Anthropic) para las tareas que lo requieren.

2.8 Disponibilidad del código

El código fuente completo del prototipo se encuentra disponible públicamente en https://github.com/ccaldera/keag-poc-paper bajo licencia abierta. El repositorio incluye el framework de evaluación con las 35 pruebas automatizadas, los scripts de inicialización y despliegue, y los detalles técnicos más específicos no cubiertos en este artículo, como valores de configuración, umbrales de similaridad e hiperparámetros del pipeline de extracción y recuperación.

III. RESULTADOS Y DISCUSIÓN

En esta sección se presentan los resultados obtenidos del framework de evaluación descrito en la sección anterior, se compara el sistema con las alternativas más relevantes en la literatura, y se documentan las limitaciones identificadas.

3.1 Resultados de evaluación

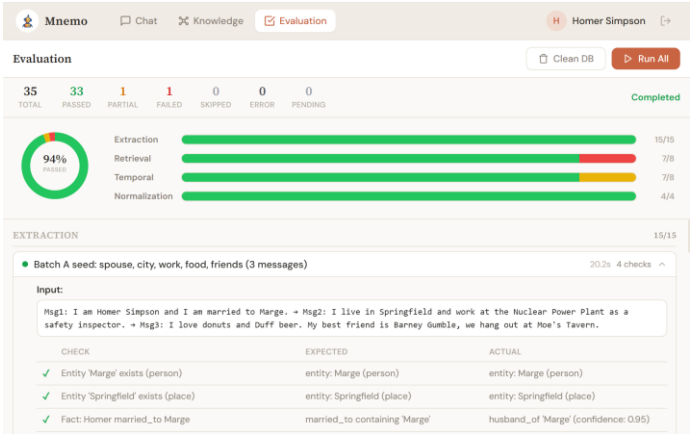
Se ejecutó el framework de 35 pruebas con Claude como modelo principal de extracción. Los resultados se resumen en la Tabla 4.

Tabla 4
Resultados generales de la evaluación.

Métrica	Valor
Total de pruebas	35
Aprobadas	33 (94.3%)
Parciales	1 (2.9%)
Fallidas	1 (2.9%)
Omitidas	0
Errores	0

La prueba fallida corresponde a la recuperación del lugar de trabajo: el sistema extrajo correctamente el hecho $works_at \rightarrow Nuclear\ Power\ Plant$ durante la fase de extracción, pero la búsqueda por hechos no lo localizó al consultarlo posteriormente. Esto indica una inconsistencia entre la extracción y la recuperación, no un fallo en el pipeline de extracción en sí. La prueba parcial corresponde a la coexistencia de mascotas en predicados acumulativos: al registrar una segunda mascota ("Santa's Little Helper"), el sistema la extrajo correctamente pero no retuvo la primera ("Snowball"), lo que sugiere una supersesión incorrecta en un predicado que debería ser acumulativo. En ambos casos, la varianza se concentra en comportamiento no determinista del LLM entre ejecuciones. La Figura 6 muestra una de las ejecuciones de las pruebas realizadas, mientras que la Tabla 5 desglosa los resultados por categoría.

Figura 6
Resultado de evaluación de framework.



Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

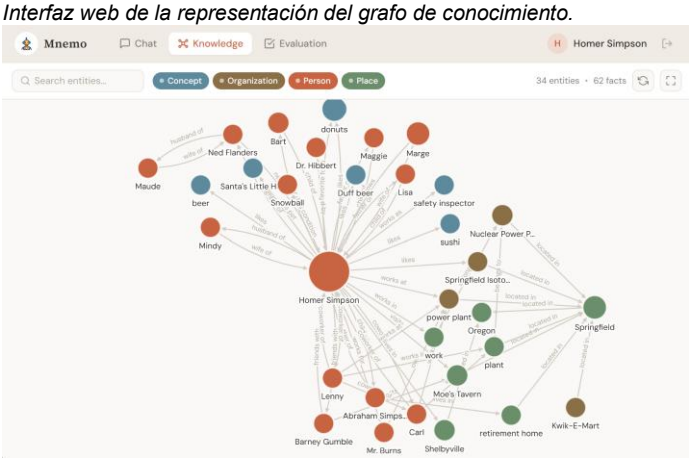
Tabla 5
Desglose de resultados por categoría de prueba.

Categoría	*Pruebas*	*Capacidad validada*
Extracción	15	Pipeline integrado, resolución de pronombres, aliases, jerarquías
Recuperación	8	Persistencia entre sesiones, búsqueda híbrida, abstención
Temporal	8	Supersesión por tipo de predicado, refuerzo sin duplicación, preservación ante mensajes irrelevantes
Normalización	4	Canonicalización de sinónimos, filtrado de efímeros, creación de inversos

Tras la ejecución de las pruebas, es posible ahora apreciar el grafo de conocimiento mediante su interfaz web, permitiendo filtrar, interactuar e ir a detalle en cada una de las entidades como se muestra en la Figura 7.

Las pruebas temporales validan escenarios específicos de evolución: actualización de edad (39 => 40 con supersesión correcta), cambio de ubicación, cambio de trabajo, refuerzo sin duplicación ("I still love donuts" no crea hechos duplicados), adición a predicados que aceptan múltiples valores ("I also like pizza" coexiste con likes => donuts), y preservación de hechos ante mensajes irrelevantes ("The weather is nice" no afecta el grafo). Las pruebas de recuperación verifican que los hechos persisten y son consultables a través de sesiones independientes, y que el sistema se abstiene correctamente cuando no tiene información.

Figura 7
Interfaz web de la representación del grafo de conocimiento.



En cuanto a costos, cada ejecución completa del framework requiere aproximadamente 31 invocaciones del pipeline (93 llamadas a la API de Claude Sonnet), con un costo estimado de $0.50 a $0.70 USD por ejecución.

3.2 Comparación con sistemas existentes

Además de la evaluación interna, se contrastaron las métricas cuantitativas reportadas por los sistemas comparables en sus respectivas publicaciones. La Tabla 6 presenta esta comparación.

Tabla 6
Comparación cuantitativa con métricas reportadas en la literatura.

Sistema	*Benchmark*	*Métrica principal*	*Valor reportado*	*Fuente*
KG-RAG Dinámico	Framework propio (35 pruebas)	Tasa de aprobación	94.3%	Este trabajo
Graphiti/Zep	Deep Memory Retrieval	Accuracy	94.8%	Rasmussen et al., 2025
MemGPT	Deep Memory Retrieval	Accuracy	93.4%	Rasmussen et al., 2025
Mem0	LOCOMO	Mejora sobre OpenAI Memory	+26%	Chhikara et al., 2025

Los benchmarks no son directamente comparables, ya que cada sistema fue evaluado en su propio framework o en benchmarks distintos. Sin embargo, la convergencia de los valores en el rango 93-95% sugiere que KG-RAG Dinámico opera en un nivel de rendimiento competitivo con sistemas del estado del arte, a pesar de ser un prototipo académico frente a productos comerciales. Los demás sistemas comparados (MemGPT, Mem0, Microsoft GraphRAG) carecen de modelo temporal explícito, lo que los posiciona en una categoría arquitectónica distinta a la propuesta de este trabajo.

La comparación de fondo se centra en Graphiti/Zep (Rasmussen et al., 2025), el sistema más cercano arquitectónicamente. La Tabla 7 resume las diferencias principales.

Tabla 7
Comparación arquitectónica.

Característica	*KG-RAG Dinámico*	*Graphiti/Zep*
Almacenamiento	PostgreSQL unificado (relacional + vectorial + FTS)	Neo4j + almacenamiento vectorial separado
Extracción	Pipeline descompuesto async (3 pasos)	Paso único de LLM
Inyección de contexto	Pre-fetch obligatorio	Tool-based
Normalización de predicados	EDC (56 predicados, matching por definición)	No documentada
Evaluación	35 pruebas automatizadas, código abierto	Benchmark propietario

Almacenamiento unificado. Graphiti/Zep requiere Neo4j como motor de grafos (con soporte adicional para FalkorDB, Kuzu y Amazon Neptune) junto con almacenamiento vectorial separado. KG-RAG Dinámico unifica grafo relacional, búsqueda vectorial (pgvector con HNSW), búsqueda léxica (FTS nativo) y deduplicación (pg_trgm) en una sola instancia de PostgreSQL. Esto reduce la complejidad operativa y garantiza consistencia transaccional: los hechos extraídos se persisten atómicamente en una sola transacción.

Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

Pipeline descompuesto. Graphiti/Zep extrae entidades y relaciones sin una descomposición explícita en subtareas. KG-RAG Dinámico separa la extracción en tres pasos especializados (entidades, hechos, jerarquías) con normalización entre cada paso, siguiendo los principios de descomposición de prompts (Khot et al., 2023). Esta separación resolvió errores frecuentes de resolución de pronombres y clasificación de entidades que se observaban con un prompt unificado.

Pre-fetch obligatorio. Graphiti/Zep depende de que el agente invoque herramientas para acceder a la memoria. KG-RAG Dinámico inyecta el contexto recuperado antes de la generación, sin depender del comportamiento del modelo. El Berkeley Function Calling Leaderboard (Patil et al., 2025) documenta tasas significativas de omisión de herramientas en escenarios multi-turno, lo que valida esta decisión.

Normalización de predicados. Graphiti/Zep no documenta un mecanismo para manejar la proliferación de predicados. KG-RAG Dinámico implementa el pipeline EDC con 56 predicados canónicos y matching semántico por embeddings de definiciones, no de labels, lo que captura la intención semántica del predicado y aborda el problema de canonicalización de predicados abiertos en grafos de conocimiento (Vashishth et al., 2018).

Evaluación reproducible. Graphiti/Zep reporta métricas en un benchmark propietario no replicable externamente. KG-RAG Dinámico incluye un framework de 35 pruebas automatizadas integrado en la aplicación, con verificación fuzzy de predicados y código fuente disponible para su reproducción.

3.3 Evaluación de alternativas de inferencia local

Para evaluar la viabilidad de sustituir Claude Sonnet y Opus por un modelo abierto ejecutado localmente, y así reducir la dependencia de APIs externas, se implementó una migración completa del pipeline utilizando Gemma 4 E4B (4B parámetros efectivos, instruction-tuned, con soporte nativo para function calling) (Gemma Team, 2026) servido mediante Ollama con API compatible con OpenAI. La migración se integró de forma limpia con la arquitectura existente, sin modificar los prompts ni la lógica del pipeline de extracción.

El experimento se ejecutó sobre 15 escenarios del framework de evaluación con hardware NVIDIA GeForce RTX 5050 (8 GB VRAM) y 64 GB de memoria RAM a 4800 MT/s en RAM. La Tabla 8 resume los resultados.

Tabla 8
Comparación de configuraciones de inferencia sobre el framework de evaluación.

Configuración	*Tasa de aprobación*
Claude Sonnet (línea base)	94%
Gemma 4 E4B	54%
Gemma 4 E4B con duplicación de prompt	63%

La técnica de duplicación de prompt consiste en enviar el contenido del mensaje dos veces dentro de la misma petición, con instrucciones explícitas de generar una sola respuesta. Esta técnica recuperó 9 puntos absolutos sin modificar el modelo, pero no cerró la brecha de 31 puntos restantes frente a Claude Sonnet. El análisis cualitativo de los fallos reveló tres categorías que exceden las mitigaciones por prompt engineering: inversión de dirección en predicados relacionales (el modelo confundía *husband_of* con *wife_of*, o *father_of* con *child_of*), alucinación de hechos sin fundamento en el texto de entrada, y generación de cadenas de predicados que violaban el esquema (espacios en lugar de guiones bajos, nombres de entidades filtrados al campo del predicado). Estas categorías de fallo corresponden a limitaciones de capacidad inherentes a modelos de 4B parámetros para un pipeline cuyo paso más exigente, la clasificación de predicados mediante EDC, ya requería razonamiento de nivel Opus con Claude.

Se concluye que, en el hardware evaluado, ningún modelo local suficientemente pequeño para ejecución eficiente alcanza la calidad necesaria para este pipeline. Modelos mayores como Gemma 4 27B MoE podrían reducir la brecha, pero no caben cómodamente en 8 GB VRAM y quedarían limitados por CPU, con pérdida de latencia inaceptable para uso interactivo. La migración hacia modelos locales permanece como trabajo futuro condicionado a la disponibilidad de modelos abiertos con capacidad demostrada para tareas de clasificación semántica profunda.

3.4 Limitaciones

Se identifican las siguientes limitaciones del prototipo actual:

Recorrido de grafo de un solo salto. La recuperación por grafo encuentra entidades y recupera sus hechos directos, pero no navega cadenas de relaciones. Una pregunta como "¿quién es la hermana de la esposa de Homero?" requeriría recorrer múltiples conexiones (Homer => married_to => Marge => sister_of => Patty). El agente puede compensar parcialmente mediante invocaciones sucesivas de herramientas, pero esto depende de que el modelo descomponga la pregunta correctamente y está limitado a 5 rondas.

Sin manejo de negación en predicados acumulativos. La declaración "ya no me gusta X" no invalida un hecho likes => X existente. La supersesión automática solo opera sobre predicados que aceptan un valor a la vez. Para predicados acumulativos, que por diseño suman valores sin reemplazar los anteriores, no existe mecanismo de remoción basado en negación en lenguaje natural.

Sin detección de contradicciones entre predicados distintos. Hechos como likes => broccoli y dislikes => broccoli pueden coexistir en el grafo sin ser detectados como contradictorios. Implementar esta detección requeriría una capa de razonamiento por restricciones fuera del alcance actual.

No determinismo en la extracción. La extracción basada en LLM es inherentemente no determinista. El mismo input puede producir predicados o clasificaciones diferentes entre ejecuciones. El framework de evaluación mide aproximadamente 94% de consistencia, con la varianza concentrada en la selección de predicados para casos ambiguos.

Dependencia de APIs externas. La calidad de la extracción de

Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

hechos y la clasificación de predicados depende de Claude Sonnet y Opus respectivamente. Cambios en estos modelos podrían afectar la consistencia del sistema. Como se documenta en la Sección 3.3, el experimento con modelos generativos locales no alcanzó la calidad necesaria; la migración hacia esa alternativa permanece como un desafío abierto.

IV. CONCLUSIONES

KG-RAG Dinámico demuestra que es viable extender el paradigma RAG con memoria persistente estructurada sin depender de múltiples motores de almacenamiento ni de intervención manual. La unificación en PostgreSQL garantiza consistencia transaccional, la descomposición del pipeline de extracción resuelve los errores de los prompts monolíticos, y la inyección obligatoria de contexto elimina la dependencia en el comportamiento probabilístico del LLM para recuperar memoria.

El modelo bitemporal es el componente con mayor impacto: preservar la historia completa de hechos sin destruir estados anteriores habilita un razonamiento temporal que los sistemas de memoria plana no pueden ofrecer. Esta propiedad es aplicable más allá de agentes conversacionales; cualquier sistema que necesite rastrear la evolución de hechos sobre entidades podría beneficiarse de esta arquitectura.

Como trabajo futuro se plantea la implementación de recorrido de grafo multi-salto, detección de comunidades para resúmenes de alto nivel, manejo de negación para predicados acumulativos, y la migración del pipeline de extracción hacia modelos generativos locales para reducir la dependencia y costo de APIs externas.

V. AGRADECIMIENTOS

Agradecimiento especial a todo el equipo de Narvalla Innovación por su ayuda a pulir la idea inicial de esta prueba de concepto y espero en un futuro próximo integrar estos hallazgos a su infraestructura tecnológica.

VI. REFERENCIAS

Anthropic. (2024). The Claude 3 model family: Opus, Sonnet, Haiku. https://www-cdn.anthropic.com/de8ba9b01c9ab7cbabf5c33b80b7bbc618857627/Model_Card_Claude_3.pdf

Chen, Z., Du, W., Zhang, W., Liu, K., Liu, J., Zheng, M., Zhuo, J., Zhang, S., Lin, D., Chen, K., & Zhao, F. (2024). T-Eval: Evaluating the Tool Utilization Capability of Large Language Models Step by Step. Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), 9510–9529. https://doi.org/10.18653/v1/2024.acl-long.515

Chhikara, P., Khant, D., Aryan, S., Singh, T., & Yadav, D. (2025). Mem0: Building production-ready AI agents with scalable long-term memory. arXiv preprint arXiv:2504.19413. https://arxiv.org/abs/2504.19413

Cormack, G. V., Clarke, C. L. A., & Büttcher, S. (2009). Reciprocal Rank Fusion Outperforms Condorcet and Individual Rank Learning Methods. Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval, 758–759. https://doi.org/10.1145/1571941.1572114

Edge, D., Trinh, H., Cheng, N., Bradley, J., Chao, A., Mody, A., Truitt, S., Metropolitansky, D., Ness, R. O., & Larson, J. (2024). From local to global: A Graph RAG approach to query-focused summarization. arXiv preprint arXiv:2404.16130. https://arxiv.org/abs/2404.16130

Gemma Team. (2026). Gemma 4 model card. Google AI for Developers. https://ai.google.dev/gemma/docs/core/model_card_4

Khot, T., Trivedi, H., Finlayson, M., Fu, Y., Richardson, K., Clark, P., & Sabharwal, A. (2023). Decomposed Prompting: A Modular Approach for Solving Complex Tasks. Proceedings of the 11th International Conference on Learning Representations (ICLR 2023). https://arxiv.org/abs/2210.02406

Kulkarni, K., & Michels, J.-E. (2012). Temporal Features in SQL:2011. ACM SIGMOD Record, 41(3), 34–43. https://doi.org/10.1145/2380776.2380786

Leblay, J., & Chekol, M. W. (2018). Towards Probabilistic Bitemporal Knowledge Graphs. Companion Proceedings of The Web Conference 2018 (WWW '18), 1–6. https://doi.org/10.1145/3184558.3191637

Luan, Y., Eisenstein, J., Toutanova, K., & Collins, M. (2021). Sparse, Dense, and Attentional Representations for Text Retrieval. Transactions of the Association for Computational Linguistics, 9, 329–345. https://doi.org/10.1162/tacl_a_00369

Packer, C., Wooders, S., Lin, K., Fang, V., Patil, S. G., Stoica, I., & Gonzalez, J. E. (2023). MemGPT: Towards LLMs as operating systems. arXiv preprint arXiv:2310.08560. https://arxiv.org/abs/2310.08560

Patil, S. G., Mao, H., Yan, F., Ji, C. C.-J., Suresh, V., Stoica, I., & Gonzalez, J. E. (2025). The Berkeley Function Calling Leaderboard (BFCL): From Tool Use to Agentic Evaluation of Large Language Models. En Proceedings of the 42nd International Conference on Machine Learning (Vol. 267, pp. 48371–48392). PMLR. https://proceedings.mlr.press/v267/patil25a.html

Rasmussen, P., Paliychuk, P., Beauvais, T., Ryan, J., & Chalef, D. (2025). Zep: A temporal knowledge graph architecture for agent memory. arXiv preprint arXiv:2501.13956. https://arxiv.org/abs/2501.13956

Reimers, N., & Gurevych, I. (2021). The curse of dense low-dimensional information retrieval for large index sizes. En C. Zong, F. Xia, W. Li, & R. Navigli (Eds.), *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)* (pp. 605–611). Association for Computational Linguistics. https://doi.org/10.18653/v1/2021.acl-short.77

Richardson, C. (2018). Microservices Patterns: With Examples in Java. Manning Publications. https://microservices.io/patterns/data/transactional-outbox.html

Snodgrass, R. T., & Ahn, I. (1985). A Taxonomy of Time in Databases. Proceedings of the 1985 ACM SIGMOD International Conference on Management of Data, 236–246. https://doi.org/10.1145/318898.318921

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., & Gurevych, I. (2021). BEIR: A Heterogeneous Benchmark for Zero-shot Evaluation of Information Retrieval Models. Proceedings of the 35th Conference on Neural Information Processing Systems (NeurIPS 2021), Datasets and Benchmarks Track. https://arxiv.org/abs/2104.08663

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

Vashishth, S., Jain, P., & Talukdar, P. (2018). CESI: Canonicalizing Open Knowledge Bases using Embeddings and Side Information. Proceedings of the 2018 World Wide Web Conference (WWW '18), 1317–1327. https://doi.org/10.1145/3178876.3186030

Zaratiana, U., Tomeh, N., Holat, P., & Charnois, T. (2024). GLiNER: Generalist Model for Named Entity Recognition using Bidirectional Transformer. Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2024). https://arxiv.org/abs/2311.08526

Zhang, Z., Dai, Q., Bo, X., Ma, C., Li, R., Chen, X., Zhu, J., Dong, Z., & Wen, J.-R. (2025). A survey on the memory mechanism of large language model-based agents. ACM Transactions on Information Systems, 43(6), Article 155. https://doi.org/10.1145/3748302

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/

No.	Rol	Descripción
1	Conceptualización [Conceptualization]	Christian Javier Caldera Sánchez
2	Curación de datos [Data curation]	Christian Javier Caldera Sánchez
3	Análisis formal [Formal analysis]	Christian Javier Caldera Sánchez
4	Obtención de fondos [Funding acquisition]	Christian Javier Caldera Sánchez
5	Investigación [Investigation]	Christian Javier Caldera Sánchez
6	Metodología [Methodology]	Christian Javier Caldera Sánchez
7	Administración del proyecto [Project administration]	Christian Javier Caldera Sánchez
8	Recursos [Resources]	Christian Javier Caldera Sánchez
9	Software [Software]	Christian Javier Caldera Sánchez
10	Supervision [Supervision]	Christian Javier Caldera Sánchez
11	Validación [Validation]	Christian Javier Caldera Sánchez
12	Visualización [Visualization]	Christian Javier Caldera Sánchez
13	Redacción - borrador original [Writing – original draft]	Christian Javier Caldera Sánchez
14	Redacción - revisión y edición [Writing – review & editing]	Christian Javier Caldera Sánchez

Doi: https://doi.org/10.62805/cid2026.CID008

This work is licensed under CC BY 4.0.
To view a copy of this license, visit
https://creativecommons.org/licenses/by/4.0/