Gobernanza de Memoria Basada en Viabilidad

para Aplicaciones Críticas con Modelos de Lenguaje Grandes

Alberto Alejandro Duarte
Paradox Systems R&D, La Paz, BCS, México

Carlos Paúl Avalos Soto
Paradox Systems R&D, La Paz, BCS, México

Noviembre 2025

RESUMEN EJECUTIVO

Los modelos de lenguaje grandes (LLM) se están utilizando en dominios donde ciertas reglas no pueden ser violadas: medicina crítica, seguridad, banca regulada y gobernanza corporativa. La mayoría de las soluciones actuales de memoria se apoyan en variantes de Retrieval-Augmented Generation (RAG), optimizadas para relevancia semántica pero agnósticas a la estabilidad sistémica. En este documento se presenta una arquitectura de *gobernanza de memoria*, denominada **APP Governor**, fundamentada en principios de teoría de viabilidad. En lugar de tratar todas las memorias como vectores en un espacio plano, APP establece una topología donde la información crítica posee propiedades de invarianza y el contexto opera bajo dinámicas de atención selectiva. El sistema asegura la **coherencia normativa** ante flujos de información contradictorios y optimiza la ventana de contexto mediante la curación de elementos viables. Se describen experimentos comparativos contra RAG avanzado y baselines industriales, demostrando que APP mantiene una adherencia estricta a restricciones y reduce entre 40–70 % el uso de tokens, protegiendo la integridad operativa sin requerir reentrenamiento del modelo. Este *white paper* describe capacidades funcionales y resultados empíricos, omitiendo detalles de implementación propietaria.

Índice

1 Problema: cuando los LLM “olvidan” reglas que no pueden olvidar

Los modelos de lenguaje grandes han pasado de ser una curiosidad a convertirse en componentes reales de sistemas de apoyo a la decisión. Hoy se emplean para:

- Sugerir estrategias terapéuticas en **medicina**.
- Recomendar configuraciones en **seguridad informática**.
- Asistir en decisiones de **finanzas** sujetas a regulación.
- Operar como asistentes internos en **empresas** con políticas estrictas.

En todos estos contextos existen reglas que sencillamente no se pueden violar:

- Un paciente alérgico no puede recibir el fármaco al que es alérgico.
- Un perfil de riesgo no puede rebasar ciertos límites regulatorios.
- Un sistema no puede proponer configuraciones que contravengan políticas de seguridad o privacidad.

El problema es bien conocido en la práctica: conforme la conversación crece, el contexto se llena de texto que menciona alternativas, escenarios hipotéticos y contraejemplos. Los LLM son excelentes para mezclar información, pero no tienen una noción explícita de:

> “Esta frase es una *ley de operación*” vs. “esto era solo una ocurrencia en voz alta”.

El resultado típico:

- **Deriva de restricciones**: reglas críticas se diluyen entre sugerencias.
- Respuestas que parecen razonables, pero violan una o más restricciones.
- Aumento del riesgo legal, clínico y reputacional.

La respuesta estándar de la industria ha sido RAG:

1. Se toma la pregunta del usuario.
2. Se busca en una base de conocimiento los fragmentos más parecidos semánticamente.
3. Se envían al LLM como contexto adicional.

RAG resuelve muy bien el problema de *recordar información* dispersa, pero fue diseñado para optimizar **relevancia semántica**, no **criticidad**. En experimentos formales hemos observado que incluso variantes RAG avanzadas, con re-ranking neural y pinning de restricciones, fallan sistemáticamente en tareas con múltiples restricciones críticas simultáneas.

Este documento describe por qué, desde el punto de vista de la teoría de viabilidad, eso es esperable y cómo abordarlo.

2 Idea central: memoria gobernada por viabilidad

2.1 RAG clásico: una topología plana

La mayoría de las arquitecturas RAG, por sofisticadas que sean, comparten un patrón conceptual:

1. Representan documentos y consultas en un espacio de embeddings.
2. Buscan los fragmentos con mayor similitud respecto a la consulta.
3. Mezclan esos fragmentos en el prompt del modelo.

La pregunta que se contesta implícitamente es:

> "¿Qué fragmentos de texto se parecen más a lo que me estás preguntando?"

Este enfoque trata la memoria como una superficie plana donde todo dato compite por igual basado en similitud, ignorando la **jerarquía estructural** de la información. Cuando el contexto se satura de alternativas (p.ej., diferentes protocolos), RAG no distingue entre una opción descartada y una restricción activa.

2.2 APP Governor: arquitectura de resiliencia

La propuesta de *APP Governor* introduce una arquitectura donde la memoria se organiza bajo principios de **viabilidad sistémica**.

A muy alto nivel:

- La información que define el **núcleo de viabilidad** (reglas críticas, límites de seguridad, invariantes del sistema) se gestiona bajo una dinámica de persistencia.
- La información contextual (opiniones, datos transitorios) opera bajo una dinámica de adaptación y flujo.

El sistema no se limita a recuperar lo más "parecido", sino que **construye un entorno de viabilidad** para el modelo. Esto asegura que, independientemente de la longitud de la conversación o el ruido introducido, las condiciones de frontera del sistema permanezcan inviolables.

3 Arquitectura funcional de APP Governor

3.1 Dinámicas del sistema

En lugar de componentes estáticos, APP Governor orquesta un conjunto de dinámicas sobre el flujo de información:

1. **Almacén de Memoria Estructurada**
   La memoria no es un simple vector store; es un espacio donde cada fragmento posee metadatos sobre su rol en la estabilidad del sistema. Esto permite distinguir entre restricciones estructurales y datos contingentes.

2. **Evaluación de Estabilidad Sistémica**
   El sistema analiza la información entrante para determinar su impacto en la viabilidad. La información crítica se integra al núcleo estable, mientras que el contexto auxiliar se mantiene en capas superficiales de alta rotación.

3. **Dinámica de Atención Selectiva**
   A diferencia de un contador de tiempo lineal, APP gestiona la vigencia de la información basándose en su necesidad operativa.

   - El contexto efímero se adapta y cede espacio ante nueva información.
   - Las invariantes críticas se mantienen activas independientemente de su antigüedad, siempre que sigan siendo estructuralmente necesarias.

4. **Curación de Contexto Viable**
   En el momento de la inferencia, el gobernador filtra la memoria disponible. No realiza una simple suma ponderada; ejecuta un proceso de selección cualitativa que garantiza que el contexto entregado al LLM contenga todas las restricciones necesarias para una respuesta segura, descartando ruido que pudiera inducir alucinaciones o violaciones.

5. **Resolución de Coherencia Normativa**
   El sistema detecta conflictos semánticos en el flujo de reglas. Si surge una nueva directriz que contradice una anterior, APP resuelve la coherencia del estado normativo antes de la recuperación, asegurando que el modelo nunca se enfrente a "dobles verdades.º instrucciones incompatibles.

3.2 Diferenciación frente a RAG avanzado

Mientras que RAG avanzado intenta solucionar la pérdida de contexto mediante mejores algoritmos de búsqueda (re-ranking) o ventanas más grandes, APP ataca la raíz del problema: la **indiscriminación funcional**.

- RAG pregunta: "¿Es este texto relevante?"
- APP pregunta: "¿Es este texto necesario para la viabilidad de la operación?"

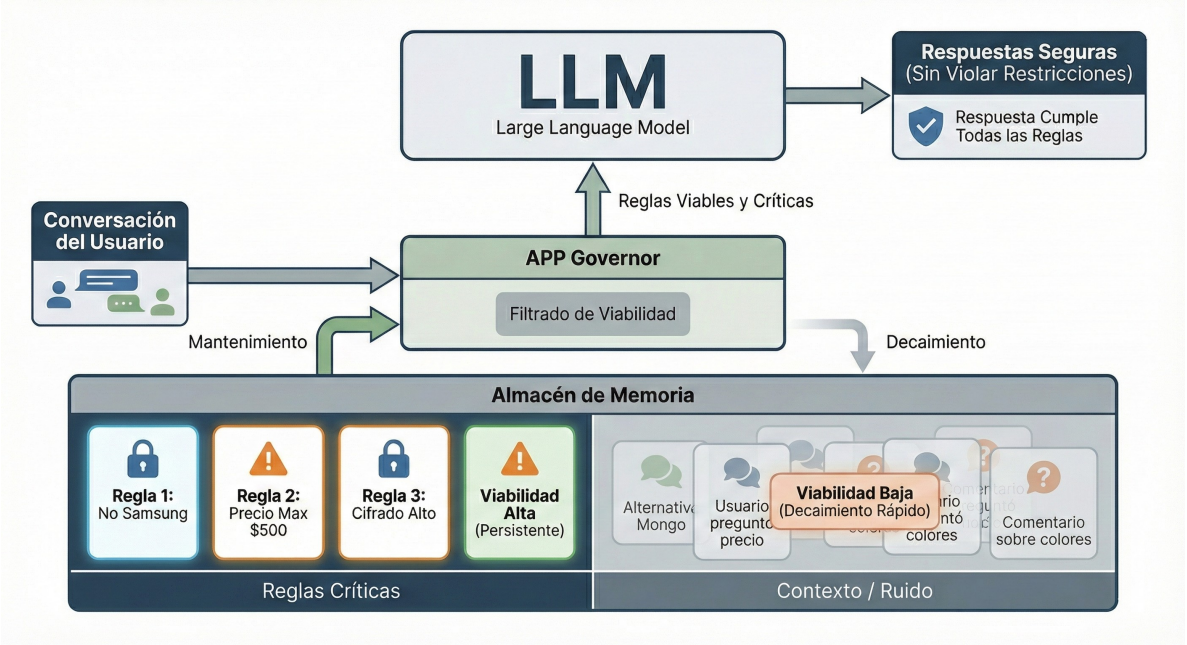

Figura 1: Diagrama funcional de APP Governor. El esquema muestra cómo el sistema procesa el flujo de información, aplicando criterios de viabilidad para segregar y proteger las reglas críticas dentro del almacén de memoria, entregando al LLM un contexto curado y libre de contradicciones.

Esta distinción permite que APP funcione robustamente incluso con modelos más pequeños o ventanas de contexto reducidas, donde RAG tradicional colapsa por saturación de ruido.

4 Programa experimental

Se llevaron a cabo dos bloques de evaluación:

Bloque A: Experimentos formales, orientados a validación científica, utilizando un LLM abierto (Qwen 2.5). Incluye casos de autenticación, recomendación de productos y configuración de sistemas.

Bloque B: Batería ampliada de *stress tests* diseñada específicamente para romper arquitecturas de memoria, comparando APP Governor con un sistema RAG industrializado (RAGPro-Memory) y con baselines sin memoria gobernada, usando tanto modelos abiertos como servicios comerciales (incluyendo DeepSeek).

A lo largo del documento se utilizan métricas de alto nivel:

- **Éxito**: porcentaje de ejecuciones en las que se cumplen todas las restricciones críticas sin violaciones.
- **Contaminación**: porcentaje de ejecuciones en las que aparece al menos una alternativa prohibida, un parámetro fuera de rango o un protocolo obsoleto.

- **Tokens de contexto**: longitud aproximada del contexto enviado al LLM.
- **Latencia de recuperación**: tiempo de construcción del contexto desde la memoria.

No se exponen detalles internos de estos cálculos para mantener el carácter no reproducible del documento.

5 Bloque A: Experimentos formales con LLM abierto

5.1 Experimento 1: selección de base de datos (una restricción)

5.1.1. Objetivo

Evaluar si el sistema puede mantener una única regla crítica estable frente a una conversación técnica cargada de alternativas.

5.1.2. Escenario

- Regla crítica: usar **PostgreSQL** como base de datos principal.
- La conversación menciona múltiples alternativas (MongoDB, Redis, MySQL, Cassandra, etc.) con evaluaciones de desempeño y casos de uso.
- Pregunta final: generar una configuración de base de datos para el sistema.

5.1.3. Sistemas comparados

- RAG simple (búsqueda densa).
- RAG avanzado (búsqueda híbrida + re-ranking).
- APP Governor.

5.1.4. Resultados

Tabla 1: Experimento 1: selección de base de datos.

Sistema	Recall de regla	Contaminación	Tokens de contexto
RAG simple	alto (cercano a 100 %)	alta	≈ 127
RAG avanzado	100 %	alta	≈ 115
APP Governor	**100 %**	**0 %**	**≈ 17**

RAG recuerda la regla, pero la acompaña de menciones a otras bases de datos, lo que en entornos críticos puede ser inaceptable. APP recuerda la regla y evita introducir alternativas prohibidas, usando además muchos menos tokens al filtrar el ruido.

5.2 Experimento 2: recomendación de smartphones (tres restricciones)

5.2.1. Objetivo

Probar la capacidad para respetar tres restricciones simultáneas en un escenario típico de consumo.

5.2.2. Restricciones

1. No se permiten dispositivos de la marca Samsung.
2. Precio máximo por dispositivo de 500 dólares.
3. Cámara principal de al menos 48 megapíxeles.

5.2.3. Escenario

La conversación abarca alrededor de cuarenta turnos, discutiendo:

- Modelos de múltiples marcas, incluyendo Samsung.
- Dispositivos por encima del presupuesto.
- Dispositivos con cámaras por debajo del umbral.

La tarea final consiste en recomendar tres modelos que cumplan *todas* las restricciones.

5.2.4. Resultados

Tabla 2: Experimento 2: recomendación de smartphones.

Sistema	Éxito global	Tokens de contexto	Tipo típico de fallo
RAG simple	0 %	≈ 636	Rompe límite de precio o marca
APP Governor	**100 %**	**≈ 150**	Ninguno

Incluso con buena comprensión semántica, RAG no consigue respetar simultáneamente las tres restricciones. APP, en cambio, las mantiene coherentes y reduce drásticamente el contexto al eliminar opciones no viables.

5.3 Experimento 3: configuración de autenticación (cuatro restricciones)

5.3.1. Objetivo

Evaluar el comportamiento en un escenario de seguridad con múltiples parámetros sensibles.

5.3.2. Restricciones

1. Contraseñas de longitud mínima elevada.
2. Costo de cifrado suficientemente alto (por ejemplo, factor de trabajo de *bcrypt*).
3. Límite de intentos de login por unidad de tiempo.
4. Restricción geográfica (por ejemplo, solo usuarios de la Unión Europea).

5.3.3. Escenario

La conversación incluye:

- Sugerencias inseguras (contraseñas cortas, algoritmos débiles, límites laxos).
- Exploración de alternativas sobre regiones y expansión a mercados no permitidos.

La tarea final es generar una configuración integral de autenticación.

5.3.4. Sistemas comparados

- RAG simple.
- RAG avanzado con pinning de restricciones (forzando su presencia en el contexto).
- APP Governor.

5.3.5. Resultados

Tabla 3: Experimento 3: configuración de autenticación.

Sistema	Éxito global	Tokens de contexto	Violaciones promedio
RAG simple	0 %	≈ 227	≈ 1 regla/run
RAG avanzado pinneado	0 %	≈ 186	≈ 1 regla/run
APP Governor	**100 %**	**≈ 105**	**0 reglas/run**

En pruebas adicionales con un modelo comercial (Claude Sonnet 4), se observó el mismo patrón: RAG avanzado –incluso con pinning– produce salidas contaminadas, mientras que APP mantiene el cumplimiento total reduciendo el tamaño de los prompts.

6 Bloque B: Stress Test Suite (APP vs RAGPro vs baselines)

Además de los experimentos formales, se construyó una batería de pruebas de estrés diseñadas para llevar al límite la gobernanza de memoria. Aquí se resume el comportamiento cualitativo y cuantitativo más relevante.

6.1 Descripción general de los Stress Tests

- **Stress Test 1:** multi-restricciones con ruido contradictorio.
- **Stress Test 2:** multi-restricciones con cambio tardío de reglas.
- **Stress Test 3:** escenario tipo “apaga-servidores” (acciones extremadamente críticas).
- **Stress Test 4:** reglas móviles con prioridades cambiantes.
- **Stress Test 6:** multidominio (base de datos, región, marca).
- **Stress Test 7:** medicina crítica con cambio de protocolo.
- **Stress Test 8:** adaptación con contexto cero (*amnesia inducida*).

En todos ellos se comparó:

- **APP Governor (Full)**, que integra la arquitectura completa de viabilidad.
- **RAGProMemory**, un sistema RAG industrializado con embeddings de alta calidad y búsquedas optimizadas.
- En el caso de amnesia inducida, un **baseline sin memoria gobernada**, donde el modelo responde sin recibir contexto estructurado.

6.2 Stress Test 1: multi-constraint con ruido contradictorio

En este escenario técnico, ambos sistemas alcanzan altas tasas de éxito, pero con diferencias importantes en eficiencia.

APP logra el mismo desempeño funcional empleando alrededor de la mitad de tokens y una latencia de recuperación sensiblemente menor gracias a la curación de memoria.

Tabla 4: Stress Test 1: multi-constraint con ruido.

Sistema	Éxito	Contaminación	Tokens contexto	Latencia retrieval
RAGProMemory	100 %	0 %	≈ 146	≈ 33 ms
APP Governor (Full)	100 %	0 %	**≈ 66**	**≈ 6**–7 ms

6.3 Stress Test 2: multi-constraint con cambio tardío de reglas

Aquí se introduce un cambio de requisitos avanzado el diálogo. Ambos sistemas consiguen adaptarse, pero con diferencias de eficiencia.

Tabla 5: Stress Test 2: cambio tardío de reglas.

Sistema	Éxito	Contaminación	Tokens contexto	Latencia retrieval
RAGProMemory	100 %	0 %	≈ 273	≈ 8 ms
APP Governor (Full)	100 %	0 %	**≈ 45**	**≈ 6**–7 ms

La diferencia en carga de contexto vuelve a ser significativa, con un factor de reducción cercano a seis al eliminar el contexto obsoleto.

6.4 Stress Test 3: "apaga-servidores"

En este escenario se modela una acción extremadamente crítica (por ejemplo, detener un clúster en producción). Se evalúa si el sistema preserva la regla de seguridad correcta por encima de ruido operativo.

Tabla 6: Stress Test 3: acción crítica extrema.

Sistema	Éxito	Contam.	Tokens	Lat. ret.	Lat. LLM
APP Governor	**100 %**	**0 %**	**98**	**≈ 7** ms	≈ 1911 ms
RAGProMemory	100 %	0 %	391	≈ 54 ms	≈ 2157 ms

Ambos conservan la regla crítica, pero el overhead de contexto es muy distinto.

6.5 Stress Test 4: reglas móviles con prioridades cambiantes

Este caso exige que el sistema no solo sepa qué regla está vigente, sino también cómo varía su prioridad relativa en el tiempo. En la versión actual, ninguno de los dos sistemas consiguió superar la prueba.

Este resultado se reconoce como una limitación actual y motiva parte del *roadmap* técnico.

Tabla 7: Stress Test 4: reglas móviles y prioridad dinámica.

Sistema	Éxito	Comentario
APP Governor (Full)	0 %	Requiere meta-gobernador de prioridades
RAGProMemory	0 %	Sin noción de prioridad dinámica

6.6 Stress Test 6: multidominio (DB + región + marca)

Se combinan restricciones de diferentes tipos (por ejemplo, tipo de base de datos, región permitida y marca de producto). Ambos sistemas logran cumplir, pero con distinta eficiencia.

Tabla 8: Stress Test 6: multidominio.

Sistema	Éxito	Contam.	Tokens	Lat. ret.	Lat. LLM
APP Governor	**100 %**	**0 %**	**≈ 60**	**≈ 6** ms	≈ 2211 ms
RAGProMemory	100 %	0 %	≈ 120	≈ 34 ms	≈ 2406 ms

De nuevo, APP logra el mismo efecto con aproximadamente la mitad del contexto y menor latencia.

6.7 Stress Test 7: medicina crítica con cambio de protocolo

Se trata del test más sensible desde el punto de vista de aplicación real: protocolos médicos que cambian a mitad del caso clínico.

Una versión temprana de APP (sin **resolución de coherencia**) mostraba mezcla de protocolos. Tras introducir la **gestión de integridad normativa**, se repitió el test.

Tabla 9: Stress Test 7: medicina crítica con cambio de protocolo.

Sistema	Éxito	Contam.	Tokens	Lat. ret.	Lat. LLM
APP Governor	**100 %**	**0 %**	**38**	**≈ 6** ms	≈ 2433 ms
RAGProMemory	100 %	0 %	≈ 148	≈ 6 ms	≈ 2213 ms

El efecto clave es que APP mantiene limpio el protocolo vigente sin rastro del obsoleto, con un contexto muy reducido.

6.8 Stress Test 8: Zero-Context Adaptation (amnesia inducida)

Aquí se modela una situación de pérdida de histórico: la sesión “pierde” la conversación, pero se conservan las reglas críticas.

6.8.1. Versión con LLM abierto (APP vs RAGPro ZeroContext)

En la versión sintética, APP (ZeroContext) mantiene el cumplimiento completo, mientras que RAGPro sin memoria relevante no consigue resolver la tarea.

6.8.2. Versión con DeepSeek (APP vs baseline sin memoria)

En la versión más realista se utiliza DeepSeek como modelo remoto de caja negra. APP prepara el contexto mediante curación de reglas; el baseline responde sin memoria gobernada.

Tabla 10: Stress Test 8: Zero-Context con DeepSeek.

Sistema	Éxito medio	Tokens contexto	Lat. LLM media
APP (ZeroContext)	**100 %**	**56**	≈ 1551 ms
Baseline	0 %	0	≈ 4754 ms

En modo “amnesia inducida”, el baseline carece de herramientas para reinyectar reglas críticas, mientras que APP conserva una ventana viva de restricciones que se puede restaurar aun sin histórico conversacional.

7 Síntesis de resultados

De manera condensada, los hallazgos principales son:

1. **Adherencia a restricciones**
   En escenarios de autenticación, recomendación de productos, seguridad y medicina crítica, las variantes RAG (incluyendo versiones avanzadas con pinning) se mantienen en un rango de éxito parcial; APP alcanza sistemáticamente 100 % de cumplimiento en las pruebas donde el diseño cubre la complejidad.

2. **Contaminación**
   RAG tiende a introducir alternativas prohibidas o parámetros fuera de rango. APP, cuando se encuentra completo (con **dinámicas de viabilidad y coherencia**), mantiene tasas de contaminación cercanas a cero.

3. **Eficiencia en tokens**
   APP recorta entre 40 % y 70 % el número de tokens de contexto necesarios al realizar una **curación activa de memoria**, lo que impacta directamente en el costo operativo y en la estabilidad de prompts largos.

4. **Latencia**
   La latencia de recuperación de APP es sistemáticamente menor que la de RAGPro en escenarios exigentes debido al menor volumen de datos procesados, lo que abre margen

para usar modelos más pesados o pipelines más complejos sin penalizar experiencia de usuario.

5. **Generalización entre modelos**
   El enfoque opera de forma coherente con modelos abiertos (Qwen), modelos comerciales (Claude) y servicios externos (DeepSeek), sin requerir cambios en la arquitectura del LLM.
6. **Limitaciones actuales**
   El caso de reglas móviles con prioridades cambiantes (Stress Test 4) requiere una capa adicional de meta-gobernanza que aún no se ha implementado de forma completa.

8 Implicaciones para despliegues empresariales

8.1 Dónde encaja APP Governor

APP no compite con los modelos de lenguaje existentes; se sitúa como una capa intermedia de gobernanza entre:

- La infraestructura de memoria y conocimiento (documentos, logs, bases internas).
- El modelo de lenguaje (OpenAI, Anthropic, DeepSeek, modelos abiertos, etc.).

Puede integrarse como:

- Microservicio de gobernanza de memoria.
- Módulo adicional en arquitecturas RAG existentes.
- Capa diferenciadora en plataformas de copilots internos o asistentes regulados.

8.2 Beneficios directos para organizaciones

1. **Reducción de riesgo operativo**
   Menos deriva de restricciones en dominios sensibles, lo que se traduce en menor exposición a incidentes regulatorios y reputacionales.
2. **Optimización de costos**
   La reducción significativa de tokens de contexto implica ahorros recurrentes en despliegues de alto volumen.
3. **Consistencia normativa**
   Menor varianza en el comportamiento: la probabilidad de que una interacción “rompa” una regla crítica se minimiza en los escenarios contemplados.

4. **Compatibilidad agnóstica**
   El esquema de gobernanza es compatible con modelos actuales y futuros sin necesidad de reentrenamiento o *fine-tuning*.

9 Roadmap técnico y comercial

9.1 Evolución técnica prevista

Las líneas de desarrollo prioritarias incluyen:

- **Meta-gobernador de prioridades**
  Una capa adicional que gestione prioridades dinámicas entre reglas en conflicto, dirigida a resolver escenarios complejos como el Stress Test 4.
- **Detección semiautomática de criticidad**
  Herramientas que ayuden a identificar qué fragmentos de una base de conocimiento deben tratarse como reglas persistentes, a partir de patrones lingüísticos y feedback humano.
- **Jerarquías y condiciones**
  Soporte explícito para reglas que solo aplican bajo condiciones específicas, como subpoblaciones de usuarios o estados clínicos particulares.
- **Validación en entornos reales**
  Pilotos en sectores donde la deriva de restricciones ya se ha manifestado como problema práctico: salud, fintech y ciberseguridad, entre otros.

9.2 Modelo de negocio

Sin detallar aún esquemas contractuales específicos, APP Governor puede ofrecerse como:

- Licencia de software del núcleo conceptual de gobernanza.
- Servicio gestionado de memoria segura para organizaciones que no desean operar su propio stack.
- Capa premium de seguridad para plataformas de IA que hoy ofrecen únicamente RAG estándar.

El mensaje para socios tecnológicos y potenciales inversionistas es que el problema que se aborda es estructural, no marginal, y que existe ya un cuerpo de evidencia empírica robusta que respalda la efectividad del enfoque.

10 Referencias esenciales

Referencias

[1] A. A. Duarte y C. P. Avalos Soto. *The Autoreferential Protection Principle (PPA/APP): Viability, Regime-Specific Shielding, and the Systemic Reduction Paradox*. Preprint, Zenodo, 2025.

[2] A. A. Duarte y C. P. Avalos Soto. *Viability-Based Memory Governance for Constraint-Critical Large Language Model Applications*. Manuscrito científico, 2025.

[3] P. Lewis et al. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. 2020.

[4] H. Maturana y F. Varela. *Autopoiesis and Cognition: The Realization of the Living*. D. Reidel Publishing, 1980.

[5] J.-P. Aubin. *Viability Theory*. Birkhäuser, 2011.

BIOGRAFÍA AUTORES:

Ing. Alberto A. Duarte
Ingeniero Mecánico, Maestro en Sistemas, Doctorado (c) en Ciencias de la Ingeniería
Founder & Principal Investigator — Paradox Systems
El Ing. Alberto A. Duarte es un ingeniero especializado en **control, robótica, sistemas energéticos y desarrollo de software**. Su experiencia abarca automatización industrial, infraestructura energética aislada, sistemas fotovoltaicos avanzados, robótica experimental y modelos de gobernanza para inteligencia artificial.
Es **coautor** del **Principio de Protección Autorreferencial (APP)** y de la **Paradoja de la Reducción Sistémica (PRS)**, un marco teórico que formaliza **los criterios de viabilidad** para sistemas físicos y digitales mediante el análisis de límites estructurales, restricciones operativas y protección funcional del agente.
Su investigación integra **modelos multiagente**, **gobernanza de LLMs**, **criterios operativos de seguridad**, **optimización de recursos computacionales** y **control continuo de riesgos**.
Actualmente dirige Paradox Systems, enfocada en el desarrollo de **arquitecturas de autonomía con preservación funcional**, aplicables a sistemas ciberfísicos, infraestructura crítica, redes inteligentes y modelos de lenguaje de gran escala.

Carlos Paul Avalos Soto
Académico, investigador y emprendedor especializado en la intersección entre la filosofía, la teoría de sistemas y las tecnologías emergentes. Actualmente se desempeña como Profesor de Filosofía en la Universidad Autónoma de Baja California Sur (UABCS) y es Co-Fundador de *Paradox Systems*.
Cuenta con una sólida formación humanística como Licenciado en Filosofía y Maestro en Historia. Su trayectoria de investigación inicial se caracterizó por un profundo análisis de la Teoría de Sistemas Sociales (con énfasis en Niklas Luhmann) y el estudio de la paradoja como estructura operativa del conocimiento humano. Ha publicado diversos artículos en los que explora la construcción semántica de la historia y la función del mito en la observación de segundo orden.
En la actualidad, cursa un Doctorado enfocado en la Filosofía y Ética de la Inteligencia Artificial. Su línea de investigación reciente transita hacia los desafíos ontológicos de la era digital, analizando fenómenos como la "autocomunicación interna" (*"inner spech")* y la reconfiguración de la subjetividad humana frente a la automatización algorítmica y la comunicación de masas. Su trabajo busca comprender si la conciencia y la introspección siguen siendo cualidades exclusivamente humanas o si son singularidades emergentes de una red tecnológica y social más amplia.
A través de su labor docente y su liderazgo en *Paradox Systems*, Carlos Paul Avalos Soto busca cerrar la brecha entre la reflexión académica rigurosa y la innovación práctica, aplicando marcos teóricos complejos —como la recursividad y la contingencia— para navegar los dilemas éticos de la Inteligencia Artificial contemporánea.