.UBAfiuba
FACULTAD DE INGENIERÍA

Procesamiento de Lenguaje Natural III

Docentes:
Mg. Oksana Bokhonok - FIUBA
Esp. Abraham Rodriguez - FIUBA

Programa de la materia

1. RAG avanzado y personalización de soluciones
2. Seguridad. Ética y alineación de modelos con valores humanos.
3. Sistemas cognitivos y agentes autónomos.
4. Escalabilidad y Optimización.
5. Diseño de un sistema Agentic IA
6. Knowledge Graphs, Agentes cognitivos y Sistemas Multiagentes
7. **Sistemas Agentic en Producción y Repaso Final**

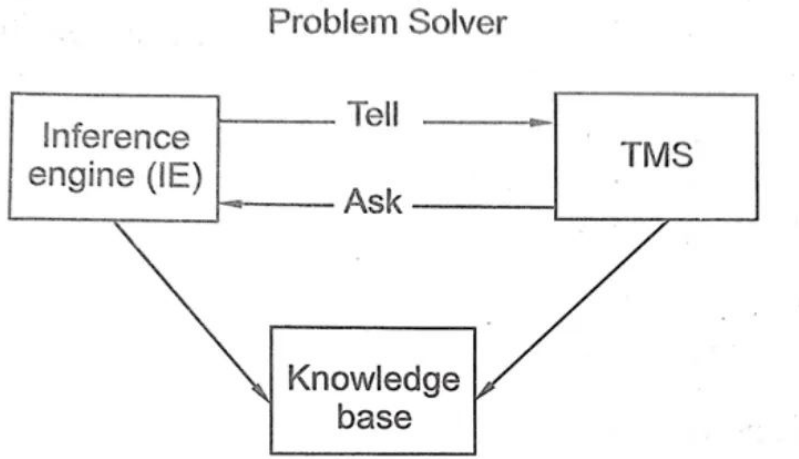
Componentes de cognición + meta-cognición

1. Bucle draft → review → execute con autocrítica calibrada.
2. Gating de acciones con políticas declarativas (policy-as-code) y umbrales adaptativos.
3. Sandboxing de herramientas (límites de syscalls/IO/tiempo) para acciones seguras.
4. Statecharts (máquinas de estados con estados anidados).
5. Sagas y compensaciones: revertir efectos secundarios tras errores del agente.
6. Planificación parcial (partial-order) para paralelizar llamadas a tools sin colisiones.
7. Generación de múltiples planes y ranking externo con evaluadores especializados.
8. **Mantenimiento de creencias (truth-maintenance): detectar contradicciones del "mundo" del agente.**
9. **Gemelos digitales / simuladores para validar políticas de acción antes de producción.**
10. **Aprendizaje online sin gradientes: ajustar reglas/heurísticas con feedback de ejecución.**
11. Parada temprana (halting) de bucles pensamiento-acción según señales de convergencia.
12. Negociación entre actores automatizados (subastas/contract-net) para asignación de tareas.
13. **Recompensas jerárquicas y objetivos múltiples (precisión, tiempo, riesgo).**
14. **Prevención de deriva de objetivos (goal drift) con "watchdogs" semánticos.**
15. Caso integrador: adjudicación de pedidos con negociación + sagas + halting.

Mantenimiento de creencias (Truth-Maintenance)

Parte de la idea de **mantener un conjunto consistente de creencias del agente**, explicar por qué cree algo y **revertir creencias cuando aparecen contradicciones**.

- **TMS clásico (Doyle):** creencias, justificativos, dependencias y *nogoods*; activa **backtracking dirigido por dependencias** cuando surge un conflicto.
- **ATMS (de Kleer):** gestiona **conjuntos de supuestos** (entornos) y calcula qué proposiciones son válidas en cada entorno; escala mejor a múltiples hipótesis.



Procedimiento:

- Guardar (hecho, soporte, supuestos).
- Propagar cambios y detectar *nogoods*.
- Explicar decisiones trazando justificativos.
- Integración con BDI/RAG: “des-creer” evidencias caducas y re-planificar.

Truth Maintenance System | by Rugvedi Ghule | Medium

A truth maintenance system - ScienceDirect

An assumption-based TMS - ScienceDirect

Gemelos digitales / simuladores para validar políticas

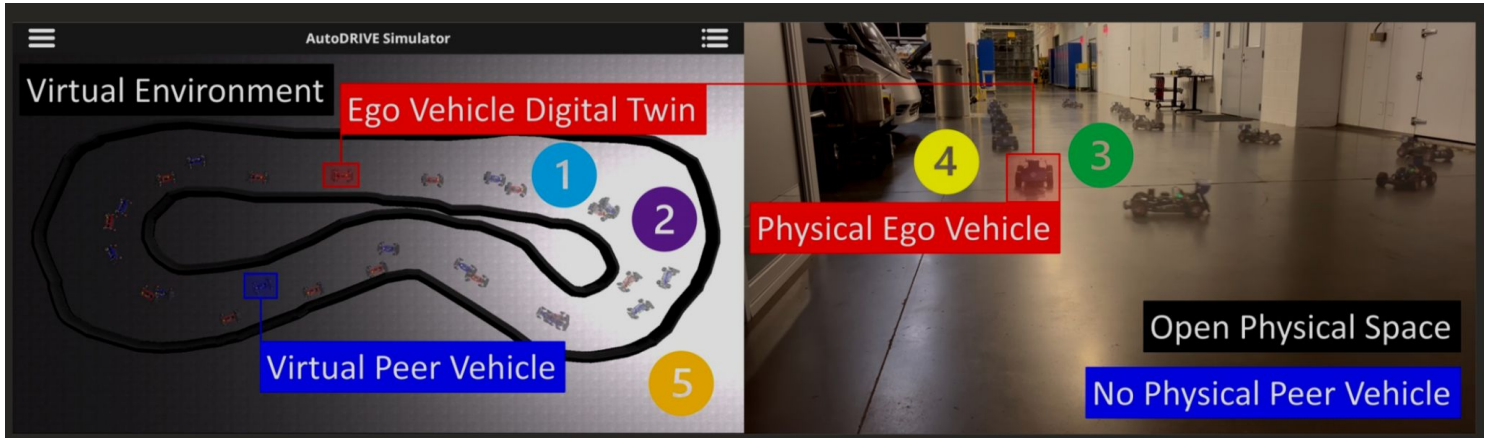
Un gemelo digital es una réplica virtual del sistema físico o lógico, que permite evaluar políticas y acciones de un agente antes del despliegue real, reduciendo riesgos y costos. *Grieves & Vickers (2002–2014)* — modelo informacional sincronizado con el activo real a lo largo de su ciclo de vida (PLM)

- **Técnicas:** *stress testing*, *domain randomization* y simulación *sim2real* con métricas de brecha para validar robustez.
- **Seguridad:** verificar **restricciones de estado, tiempo o acceso** antes de aprender o actuar (*safe RL*).
- **Buenas prácticas:** oráculos de métricas, semillas reproducibles, generadores de escenarios y *fuzzing* semántico para cubrir casos límite.

A Scalable and Parallelizable Digital Twin Framework for Sustainable Sim2Real Transition of Multi-Agent Reinforcement Learning Systems

https://www.jmlr.org/papers/volume16/garcia15a/garcia15a.pdf

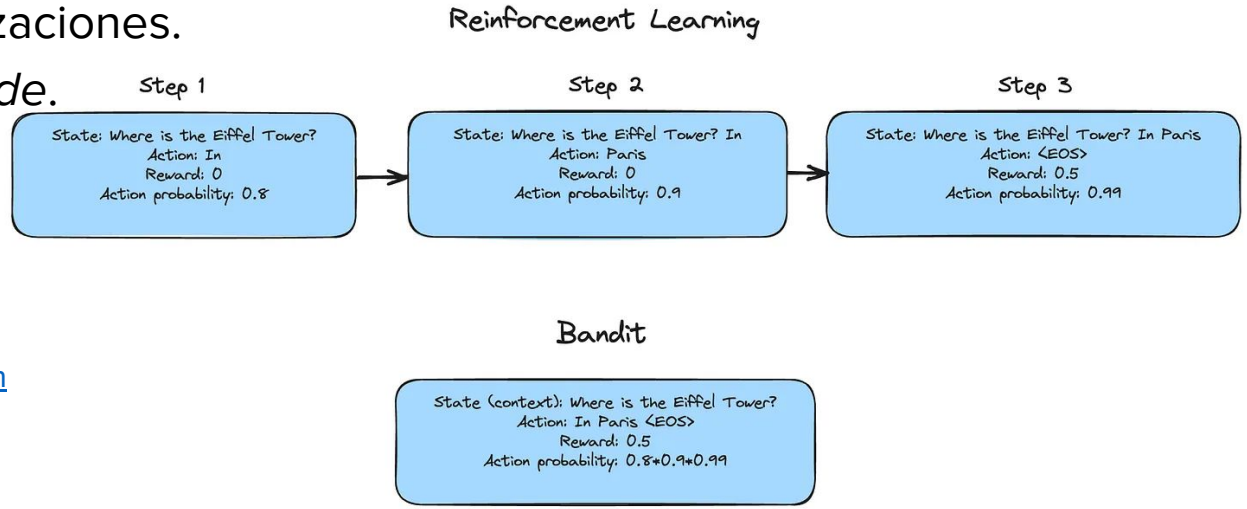
(PDF) Origins of the Digital Twin Concept



Aprendizaje online sin gradientes

Las reglas se ajustan en vivo con **feedback de ejecución** usando métodos *gradient-free*.

- **Bandits & OCO con feedback parcial:** *dueling bandits*, *bandit OCO* (“gradient descent without a gradient”): actualizar políticas de decisión con recompensas parciales/preferencias.
- **Estrategias evolutivas (ES):** optimización *black-box* de hiper-parámetros y reglas; escala masivamente en CPU, tolera horizontes largos y recompensas retrasadas.
- **Aplicación:**
  - *Bandit over tools/policies* (elegir herramienta/plantilla).
  - Tuning online de umbrales de *gating* y penalizaciones.
  - ES/CEM para ponderar reglas en *policy-as-code*.



Tangetal2020Online.pdf

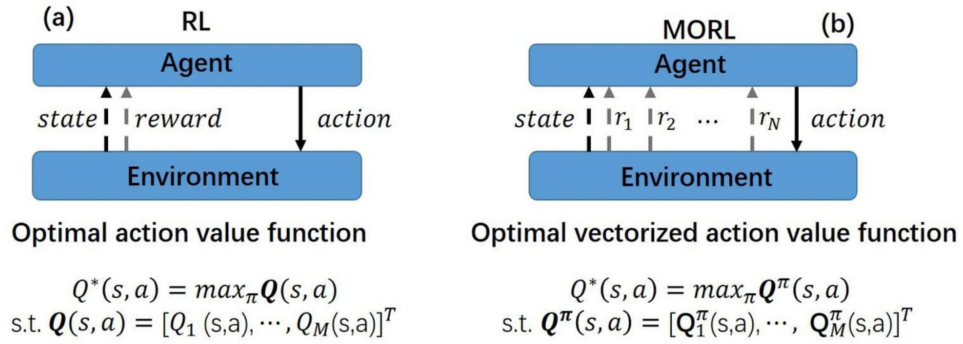
[2005.04544] Unified Models of Human Behavioral Agents in Bandits, Contextual Bandits and RL

Bandits vs Reinforcement Learning from Human Feedback

Recompensas jerárquicas y objetivos múltiples

Optimizar **varios objetivos** (precisión, tiempo, riesgo) y descomponer metas en **opciones/sub-tareas**.

- **HRL (temporal abstraction / Options):** políticas de alto nivel que invocan *opciones* (sub-políticas); acelera aprendizaje y permite **recompensas jerárquicas**.
- **MORL (multi-objective RL):**
    - **Escalarización** (suma ponderada, Chebyshev) vs **frente de Pareto** (Pareto Q-learning, conjuntos de políticas).
    - Preferencias dinámicas y restricciones (tiempo/SLAs/riesgo).
- **Diseño práctico:** define métricas canónicas, normaliza, fija **lexicografía** , y usa *constraints* para garantías duras.



Achieving optimal trade-off for student dropout prediction with multi-objective reinforcement learning [PeerJ]

1809.09332

Prevención de deriva de objetivos con watchdogs semánticos

Evitar que el agente desvíe su conducta de los objetivos/limitaciones declaradas.

- **Monitores formales (Runtime Verification):** verifican en ejecución propiedades **LTL/TLTL**; emiten alarmas o bloquean acciones.
- **Shielding en RL:** *shields* corrigen o filtran acciones que violarían especificaciones temporales; preservan seguridad durante aprendizaje/ejecución.
- **Revisión de intenciones (BDI):** política de **reconsideración** para detectar cuando cambiar metas/planes sin “deriva oportunista”.
- **Watchdogs semánticos (práctico):**
  - Lista blanca/negra de *invariantes* en NL→LTL.
  - Umbrales de **desviación semántica** (similaridad con plan/brief).
  - Acciones de mitigación: *halt*, *roll-back*, pedir confirmación o escalar a humano.

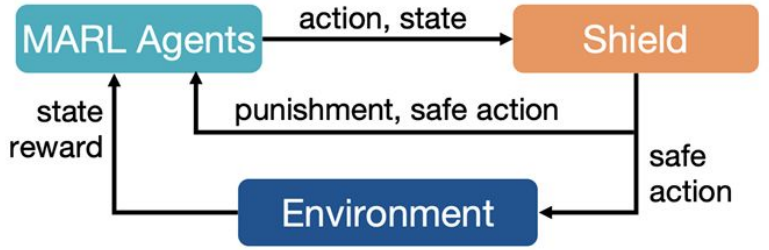


Figure 1: Safe MARL with centralized shielding.

Safe Multi-Agent Reinforcement Learning via Shielding

Optimización de costos

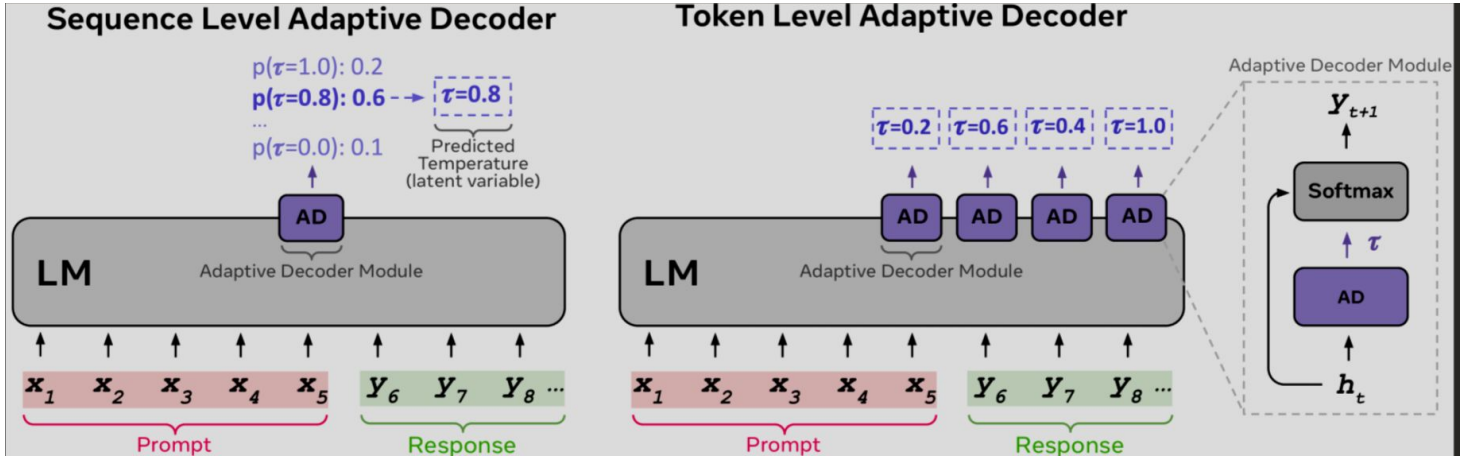
1. Speculative decoding (base+draft) para reducir latencia/costo por token.
2. Early-exit / Adaptive Computation Time en decodificación.
3. **Schedules dinámicos de temperatura/top-p por paso para minimizar tokens inútiles.**
4. Bandits contextuales para elegir modelo/proveedor por request (costo-latencia-calidad).
5. Guardrails de presupuesto y planificación cost-aware por flujo.
6. **Truncamiento y longitudes máximas adaptativas según tipo de tarea.**
7. KV-cache policies: reemplazo, compartición y compresión (mezcla 8-bit/FP16).
8. Batching elástico con colas y clases de SLA (prioridades/tiempos objetivo).
9. **Decodificación asistida tipo Medusa/EAGLE-like (árboles de candidatos con verificador).**
10. Distillation operacional: enseñar endpoints críticos a modelos chicos para abaratar.
11. **Delta prompting: enviar solo diferencias respecto del contexto previo.**
12. Arbitraje de proveedores por precio/latencia en tiempo real bajo restricciones de calidad.
13. **Serving tolerante a preempciones (spot-friendly) con reanudación de estados.**
14. **Cómputo oportunista (ventanas off-peak y scheduling consciente de tarifas).**
15. TCO y análisis de sensibilidad: escenarios y punto de equilibrio por volumen.

Schedules dinámicos de temperatura / top-p por paso

En lugar de usar temperatura o top-p fijos, el modelo ajusta estos valores **en cada paso de decodificación** según la **entropía o confianza** del contexto. Se busca reducir tokens redundantes o incoherentes sin perder creatividad (menor *token waste*, mayor *log-likelihood efficiency*.)

- ***Entropy-aware scheduling*:** temperatura alta al inicio, baja al final (p. ej., “cooling schedule”).
- ***Adaptive nucleus sampling*:** top-p ajustado por varianza de logits.

Durante reasoning loops, el planificador puede usar temperatura alta para brainstorming y baja para la acción final.

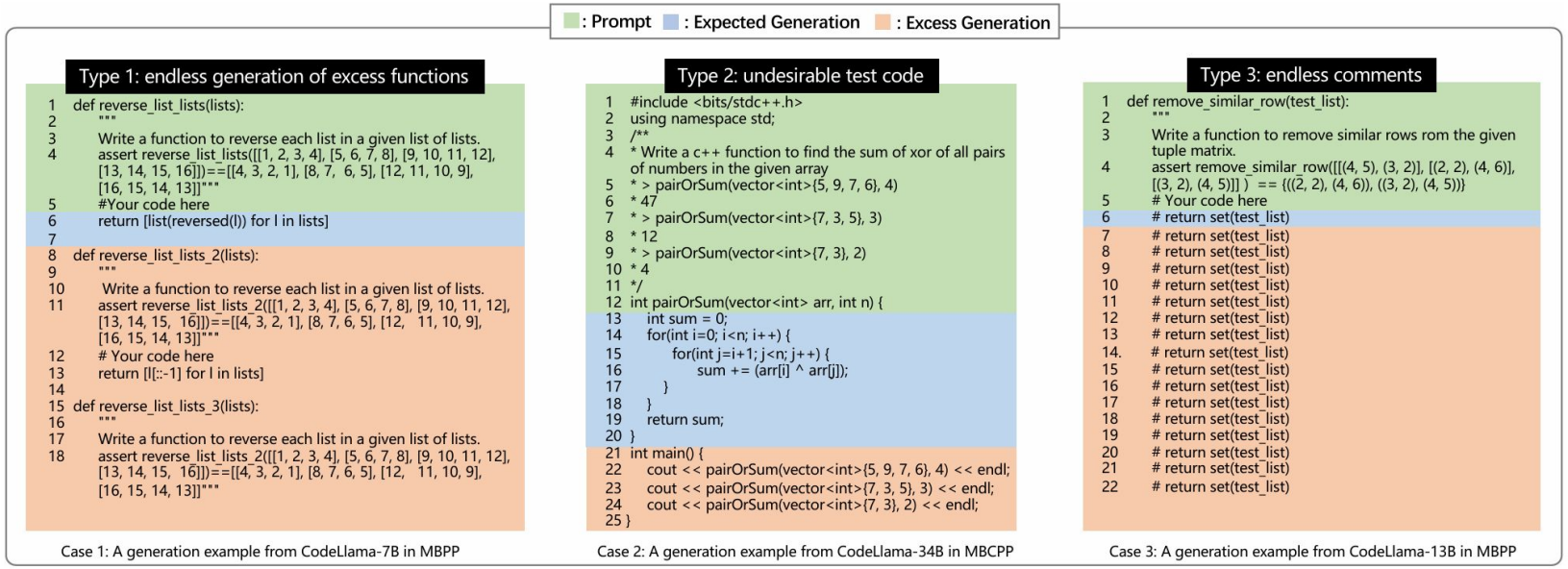


[Adaptive Decoding via Latent Preference Optimization]

Truncamiento y longitudes máximas adaptativas

Se centra en ajustar dinámicamente la **longitud máxima de salida** según el tipo de tarea, contexto o “plan length”, buscando ahorro de cómputo y menor latencia sin truncar respuestas útiles. Evita que el agente gaste tokens explicando en exceso durante sub-tareas (ej. reasoning interno).

- *Task-aware caps*: clasificación corta (≤ 20 tokens), reasoning moderado (≤ 300 tokens).
- *Entropy-based stop*: detener cuando la probabilidad marginal se estabiliza.
- *Dynamic eos prediction*: modelo auxiliar predice punto de corte óptimo.



[When to Stop? Towards Efficient Code Generation in LLMs with Excess Token Prevention]

Delta prompting

Se busca enviar al modelo **solo las diferencias** respecto del prompt anterior (Δ-contexto), aprovechando caches KV. Esto evita retransmitir miles de tokens repetidos; útil en *chat-continuations* o *reasoning loops*.

- *Incremental serving*: mantener cache activa por sesión.
- *Sparse KV updates*: solo nuevos embeddings.

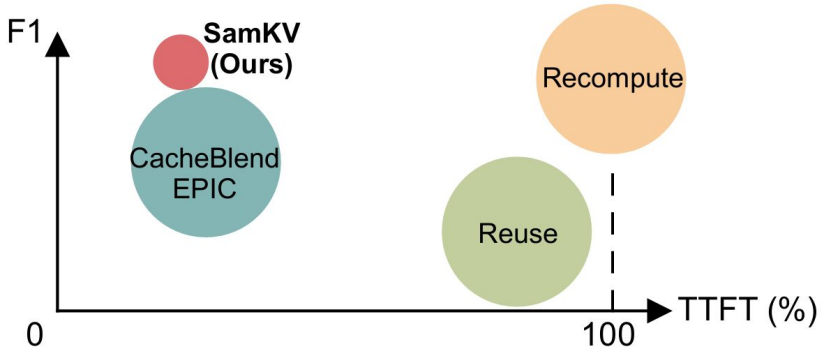


Figure 1: KV Cache methods across multiple contexts. The x-axis shows time to first token (TTFT) as a percentage of full recomputation. The y-axis represents F1 scores, with circle sizes indicating GPU memory usage during inference.

Sparse Attention across Multiple-context KV Cache

Prompt caching - OpenAI API

Prompt caching with Azure OpenAI in Azure AI Foundry Models - Azure OpenAI | Microsoft Learn

Serving tolerante a preempciones (spot-friendly)

Servidores de inferencia capaces de **reanudarse tras interrupciones** (por GPU spot preemption). Reduce costos en nubes spot / preemptibles. Permite mantener conversaciones o reasoning prolongado aunque una instancia se interrumpa.

- Checkpoints incrementales de KV cache y estado de decodificación.
- *Stateless resume* con persistencia externa (Redis, S3).
- Reasignación automática de requests.

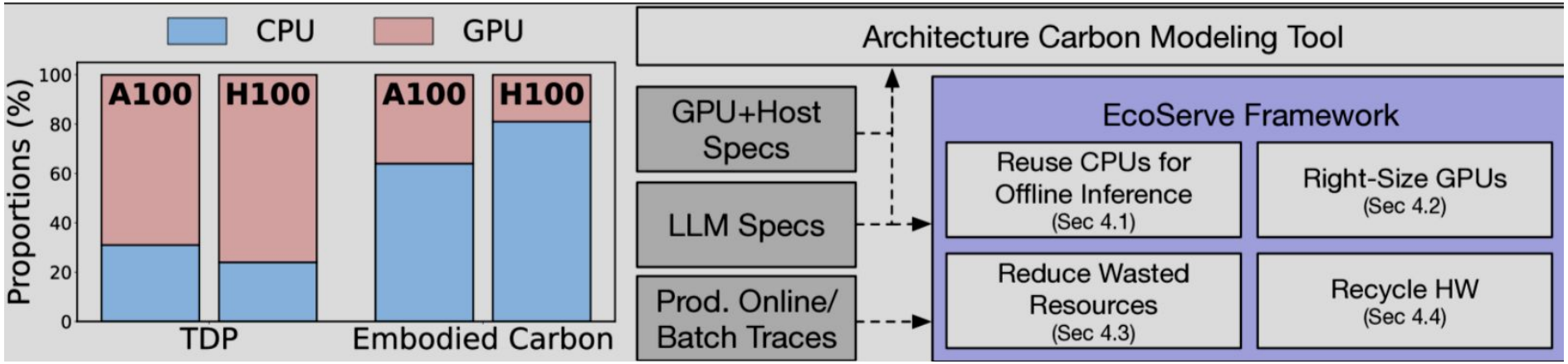
osdi24-fu.pdf

SpotServe: Serving Generative Large Language Models on Preemptible Instances

Cómputo oportunista (off-peak / tarifas conscientes)

Planificar inferencias costosas durante ventanas **off-peak** o con menor precio energético / cloud. Garantiza menor costo operativo.

- *Dynamic scheduling* según precio spot.
- *Batching elástico* con colas diferidas.
- *Demand forecasting* (pre-fetch o *warm-cache*)
- *Carbon-aware scheduling: prioriza ejecuciones en momentos y regiones con menor intensidad de carbono para reducir la huella total del sistema.*



EcoServe: Designing Carbon-Aware AI Inference Systems

Automatización code-first

1. DSL declarativo para describir políticas, herramientas y flujos (contratos explícitos).
2. **Verificación formal ligera de flujos (invariantes con TLA+/Alloy simplificado).**
3. Testing de contratos y property-based testing para herramientas de agentes.
4. **Fuzzing de parámetros (generadores de inputs + oráculos de invariantes).**
5. Replays deterministas con event-sourcing para depurar fallas no reproducibles.
6. Análisis estático/linters de prompts y de llamadas a tools para detectar anti-patrones.
7. Runners aislados (WASM/containers mínimos) con cuotas de CPU/Mem/IO por acción.
8. Despliegues prudentes: shadow/canary específicos para flujos de agente (rollback automático).

Fuzzing de parámetros

Se busca probar la robustez de un agente o modelo generando **inputs pseudoaleatorios estructurados** (fuzzing), combinados con **oráculos** que verifican si se violan reglas o invariantes.

- **Property-based testing (QuickCheck / Hypothesis):** generar datos válidos y de borde para cada herramienta del agente.
- **Input fuzzing:** explorar configuraciones extremas (umbrales, temperaturas, tamaños de contexto, latencias simuladas).
- **Oráculos:** invariantes que definen comportamiento correcto (ej.: “nunca tokenizar más de N por paso”, “respuesta ≤ 3s en modo síncrono”).
- **Coverage-guided fuzzing:** prioriza inputs que alcanzan estados nuevos del planificador o del pipeline.

```python
from hypothesis import given, strategies as st
import json

@given(st.text())
def test_agent_returns_valid_json(prompt):
    response = agent.generate(prompt)
    try:
        parsed = json.loads(response)
    except json.JSONDecodeError:
        assert False, f"Invalid JSON: {response}"
```

2408.00523

Verificación ligera + Fuzzing en el ciclo del agente (Autoverificación y Robustez Operativa)

Ambos métodos permiten **“romper” al agente antes de producción** para garantizar que sus flujos sean: seguros, recuperables, y resistentes a inputs inesperados o condiciones límite. En un ecosistema de agentes multi-tool o multi-actor, son el equivalente al **crash test de la IA**.

Paso 1 — Razonamiento y planificación: El agente genera hipótesis, planes y acciones (draft ➜ review ➜ execute). Cada plan se asocia a **invariantes declarados**: seguridad, consistencia, reversibilidad. Los invariantes se expresan en lógica simplificada tipo **TLA/ Alloy-lite**.

Paso 2 — Verificación formal ligera (Validación parcial de invariantes críticos mediante modelos formales simplificados) Antes de ejecutar, el plan pasa por un **verificador interno** que evalúa las invariantes en tiempo real, bloquea o corrige acciones fuera del dominio permitido, genera trazas de prueba. Implementación práctica: assertions o reglas declarativas (policy-as-code) compiladas a asserts en Python o Rust.

Paso 3 — Fuzzing controlado y oráculos de robustez. En paralelo, un módulo de **AI Fuzzer** introduce entradas sintéticas o variaciones extremas: prompts deformados, combinaciones de herramientas, fluctuaciones de temperatura/top-p. Cada ejecución es validada por **oráculos de invariantes** (no violar límites, no colapsar coherencia). Detecta race conditions, deadlocks, o loops en reasoning prolongado. Objetivo: garantizar que el agente siga cumpliendo las propiedades formales bajo estrés.

Paso 4 — Retroalimentación y mejora continua. Los fallos detectados por fuzzing o verificación se retroalimentan: se hace ajuste de umbrales (adaptive gating), se realiza reentrenamiento de políticas de compensación, actualización de invariantes o ampliación del modelo de flujo.

Sistemas en producción

Replit

Basado en el stack de LangChain, LangGraph y LangSmith, es un software para la creación de proyectos mediante vibe coding.

LangChain: Replit

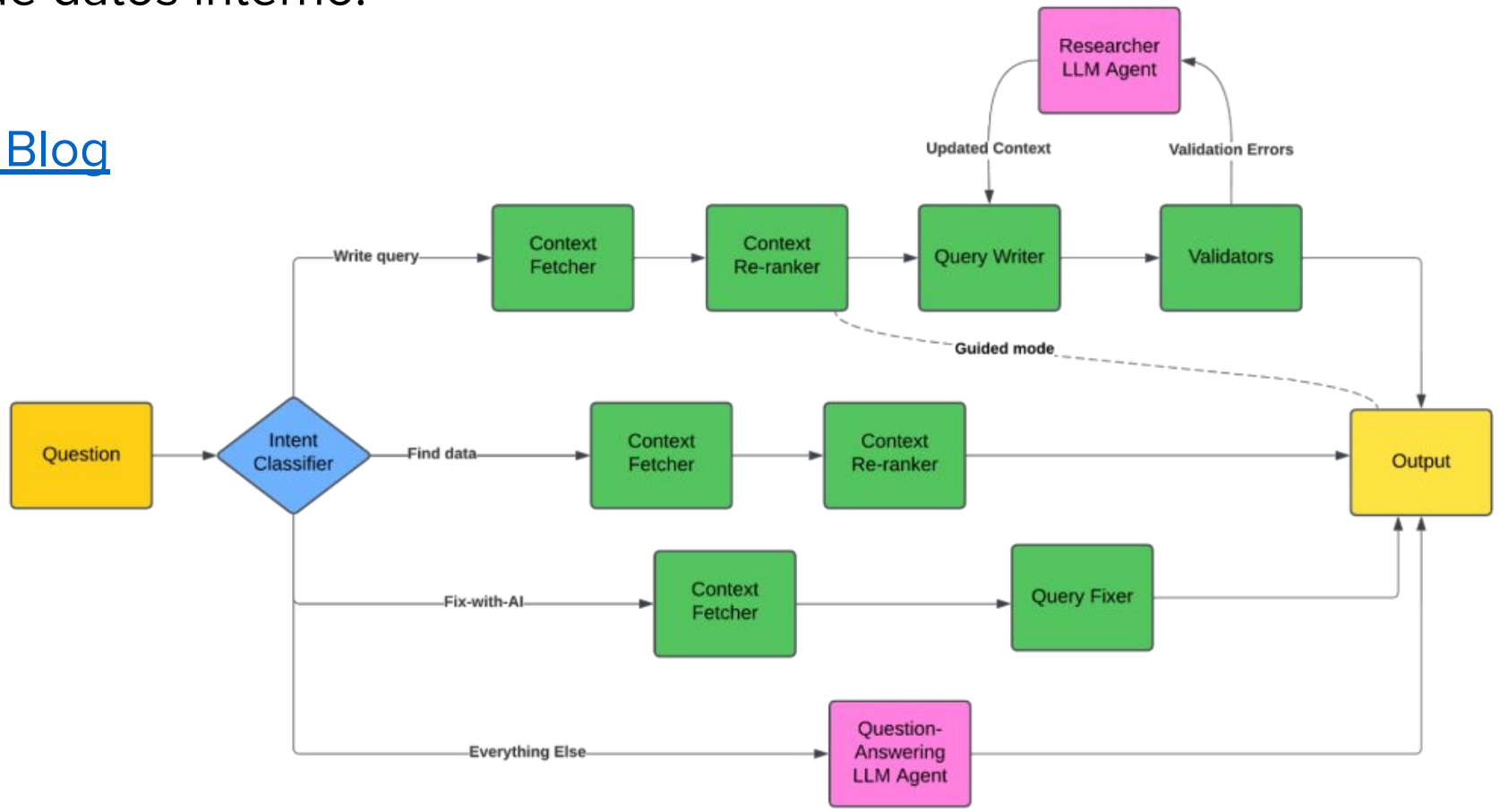
Replit Website

How We Built It: Replit Agent - Fireside Chat

LinkedIn Text-to-SQL

Basado en el stack de LangChain, LangGraph y LangSmith, LinkedIn integró un agente para análisis de datos interno.

[LinkedIn Engineering Blog]



Vercel AI SDK

Básicamente es una competencia de LangChain sin embargo, con un enfoque en desarrollo de UI para aplicaciones de AI generativa.

Puede integrarse con langchainJS o con sistemas de agentes que expongan el protocol A2A (distribuido)

Vercel AI SDK Docs

Noticias y futuros estándares

OpenAI

OpenAI ha establecido estandares a traves de todo el stack, desde el preentrenamiento y fine-tuning de GPTs hasta la API formato OpenAI.

Recientemente OpenAI ha publicado productos y protocolos que potencialmente cambiarán las aplicaciones y desarrollo de sistemas de agentes.

Agentic Commerce Protocol (ACP)

ACP es un protocolo (similar a MCP y A2A) para integrar plataformas de comercio y generar flujos de compra de manera nativa en plataformas/servicios.

Según el protocolo:

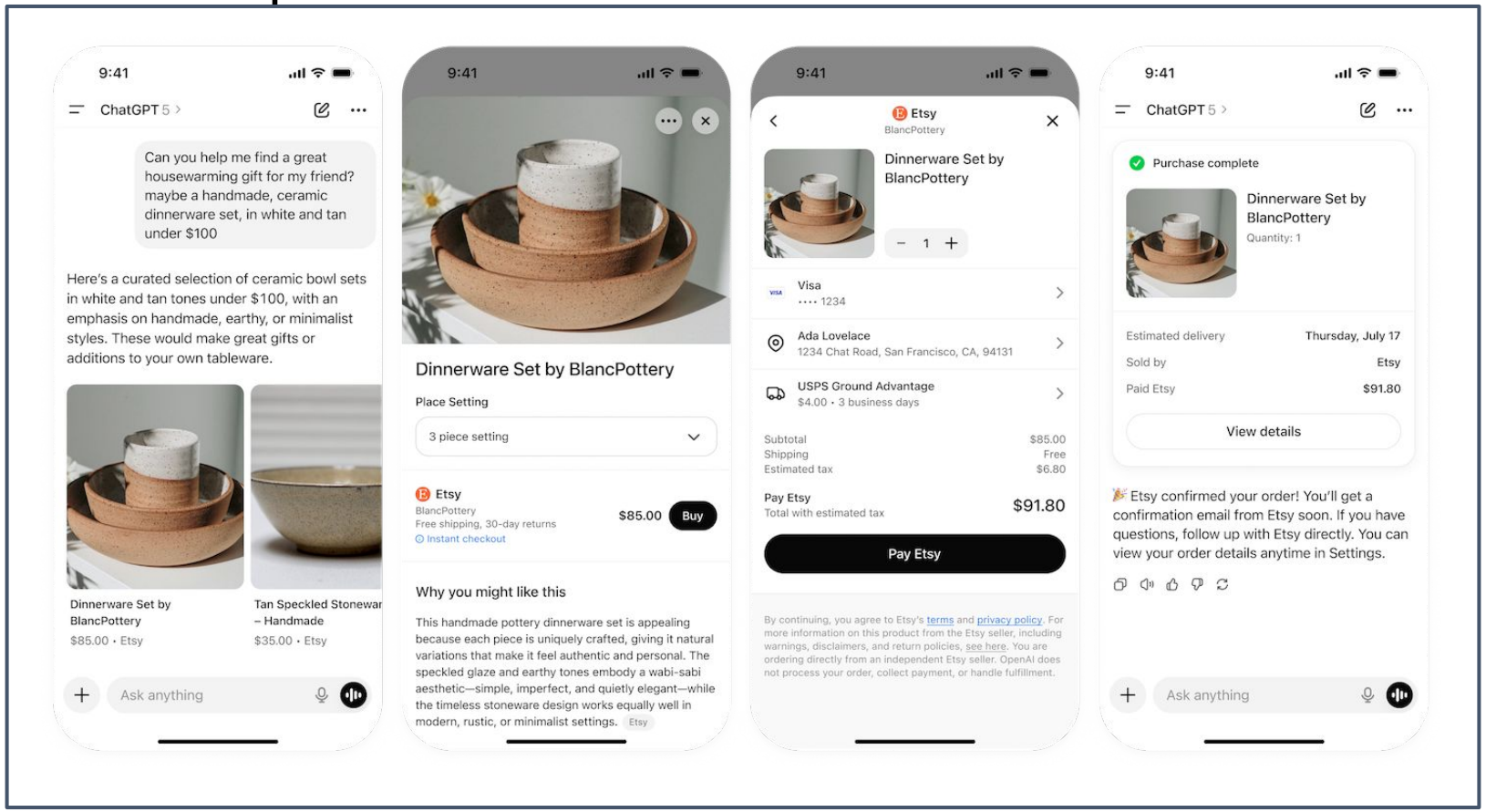
Para empresas
Llega a más clientes. Vende a compradores con alta intención de compra haciendo que tus productos y servicios estén disponibles para adquirir a **través de agentes de IA,** todo mientras utilizas tu infraestructura de comercio existente.

Para agentes de IA
Integra el comercio en tu aplicación. Permite que tus usuarios descubran y realicen transacciones directamente con las empresas dentro de tu aplicación, sin que tengas que ser el comerciante registrado.

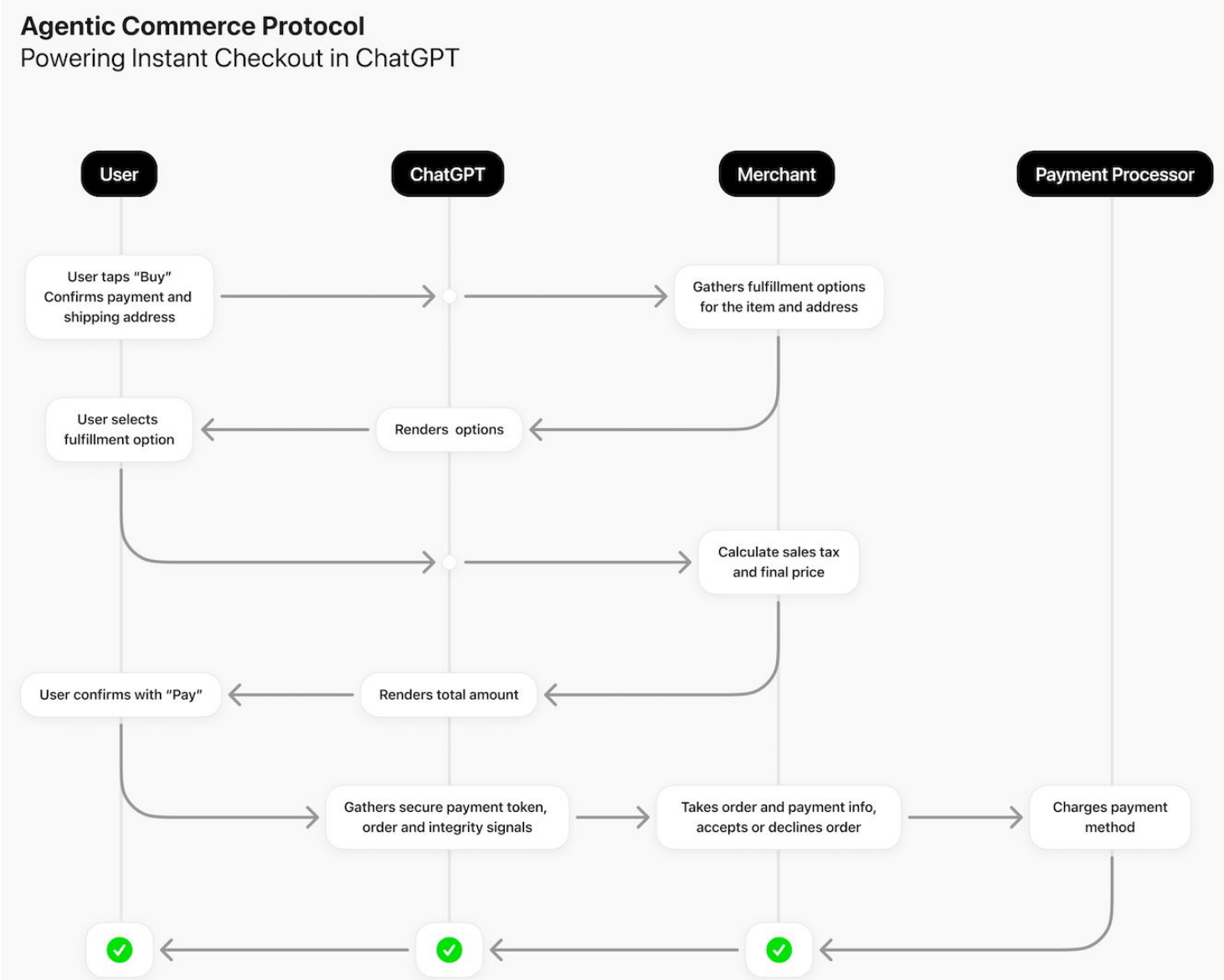
Para proveedores de pago
Aumenta tu volumen. Procesa transacciones realizadas por agentes transmitiendo tokens de pago seguros entre compradores y empresas a través de agentes de IA.

Agentic Commerce Protocol (ACP)

Implica que sistemas como ChatGPT pueden ser medios directos de ingresos a empresas de comercio.



Agentic Commerce Protocol (ACP)

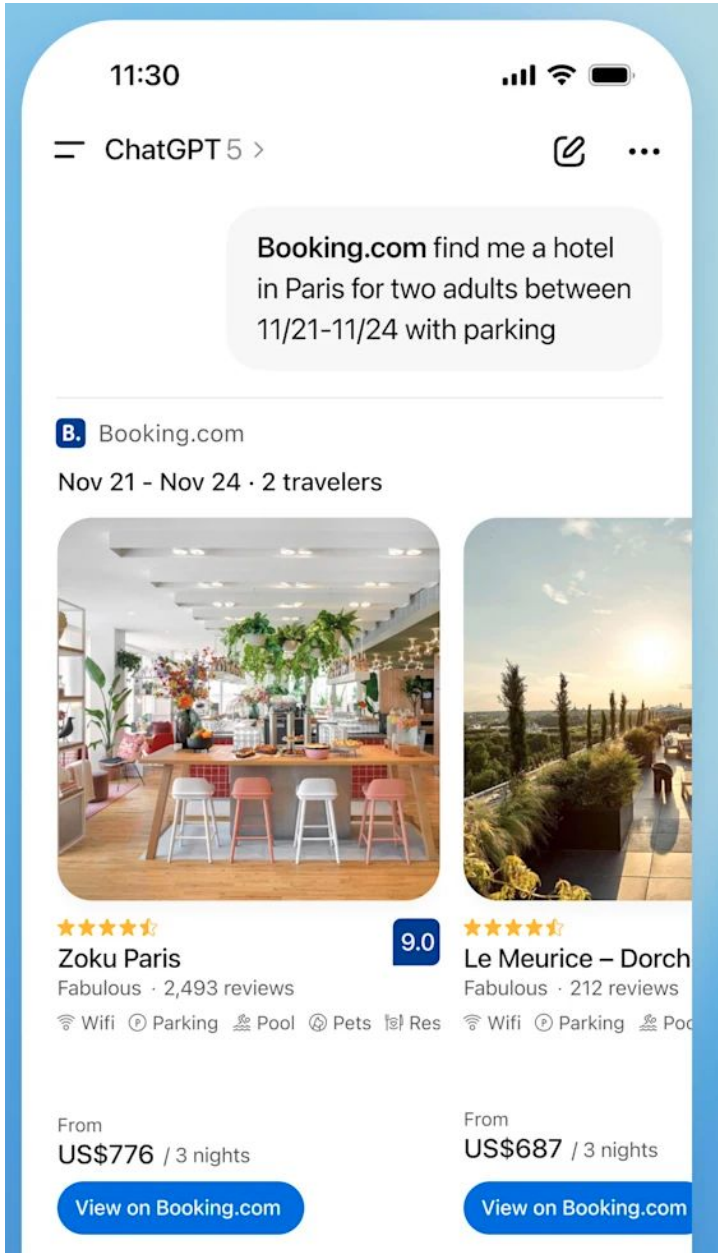


OpenAI Apps SDK

Integración con Apps de manera nativa en ChatGPT.

El SDK es una abstracción sobre MCP. Extiende a MCP para permitir definir interfaces custom de la App y la lógica de chat.

App Design Guidelines
App Developer Guidelines
MCP Server Setup
Custom UX
E2E Example



Protocolos

En resumen, actualmente los protocolos como **MCP y A2A** están siendo utilizados para crear aplicaciones nativas en plataformas utilizadas a diario (ChatGPT, Claude, Gemini) sin embargo, esto trae problemas/preguntas para otras organizaciones como:

¿Realmente vale la pena desarrollar un sistema de agentes custom?

- En especial para sistemas customer-facing como, asistentes de viaje (Expedia).

¿Es suficiente con exponer nuestros backends mediante MCP?

Recapitulando

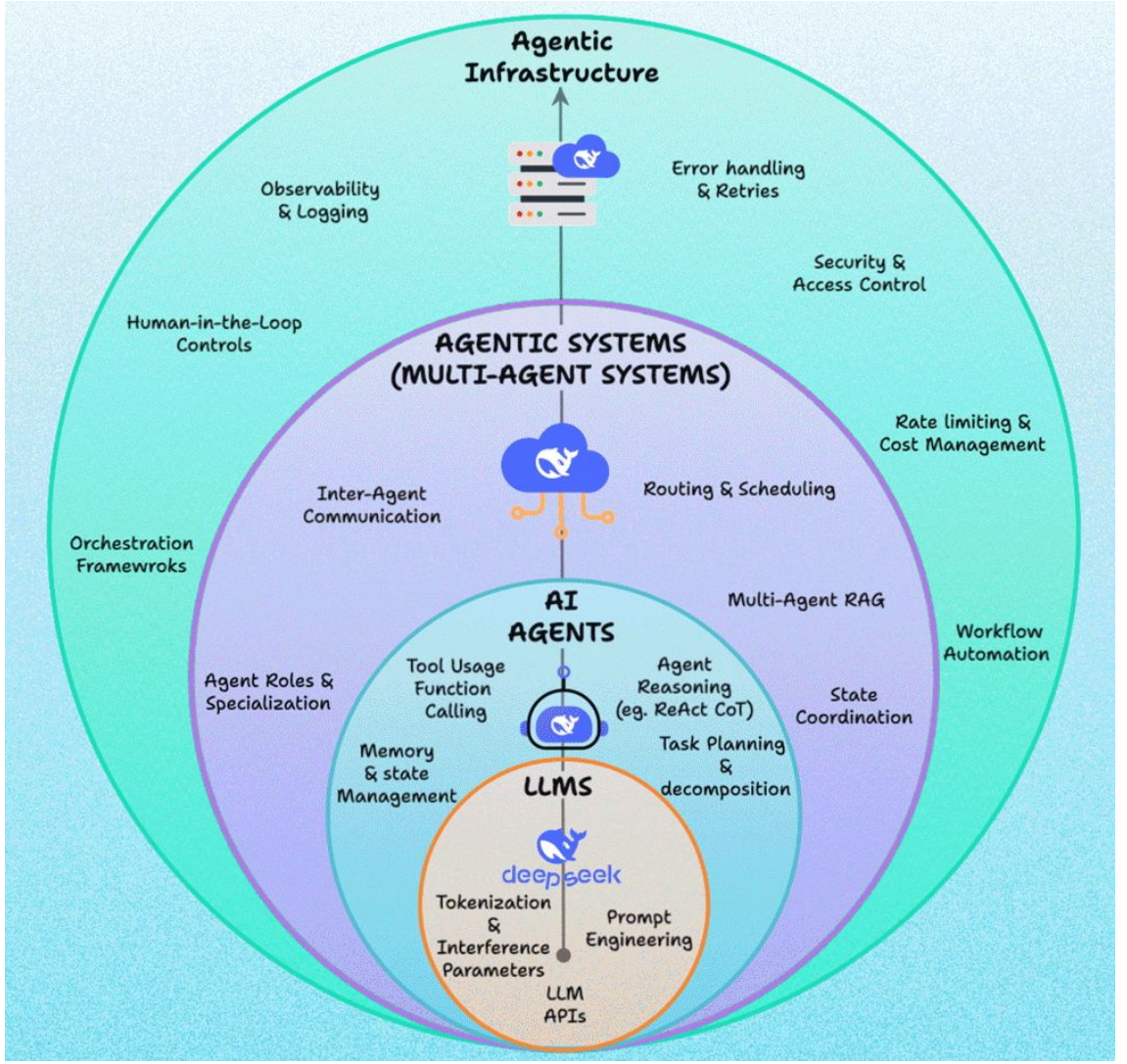
Visión por computadora III, Procesamiento de lenguaje natural I y II

Recapitulando

Empezando por los transformers vimos el stack completo:

- Pre-entrenamiento
- Fine-tuning
- Inferencia
- Aplicaciones de agentes
- Modelos multimodales.
- Entre otros.

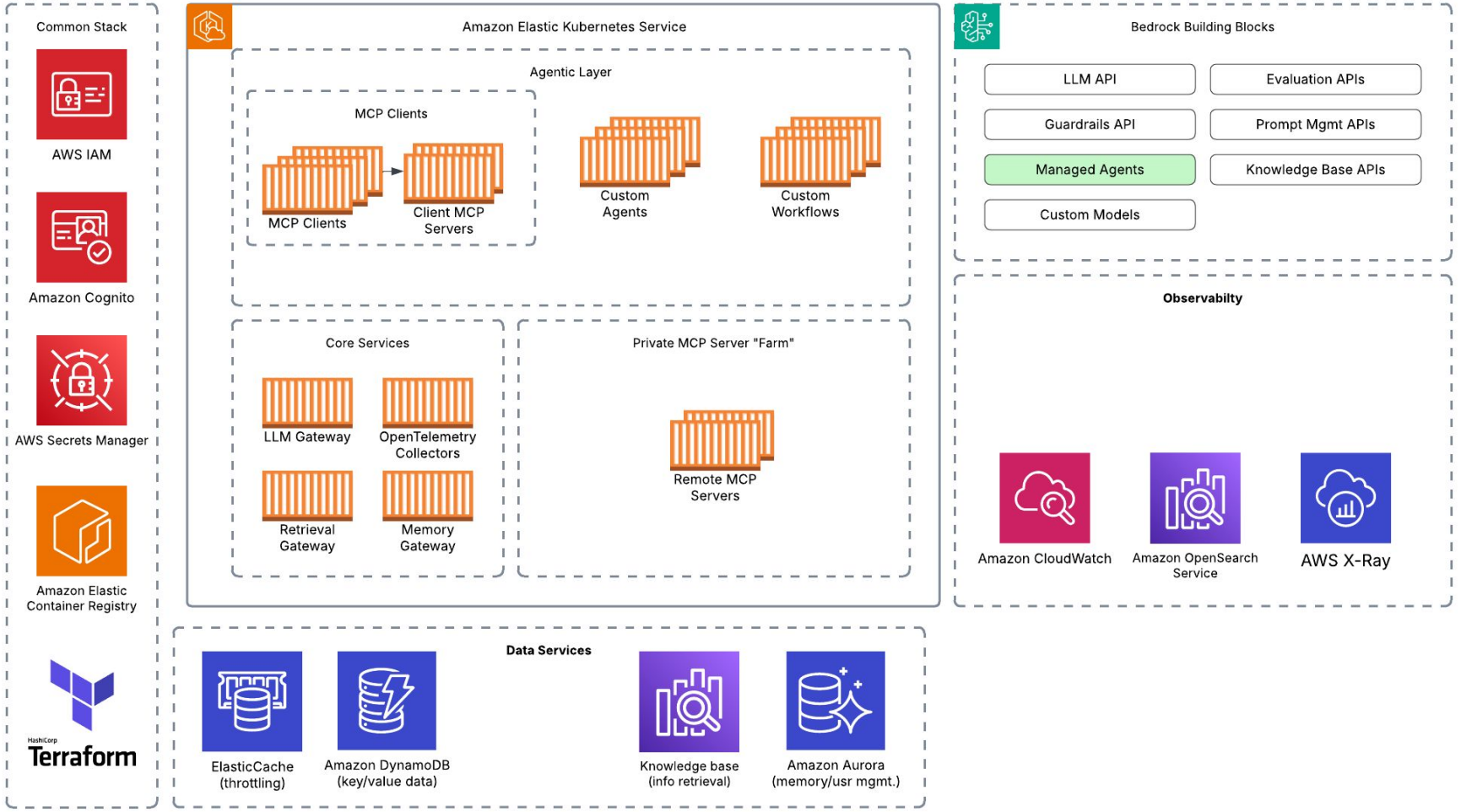
Podemos crear/fine-tune modelos y servirlos por sí solos (inference server) o crear aplicaciones de agentes.



Recapitulando

Los sistemas de agentes son complejos.

Es un momento de reflexión en especial con las integraciones que están ocurriendo hoy en dia.



Recapitulando

Existe una gran variedad de atacar problemas, las soluciones son situacionales, ejemplo:

Decidir si **fine-tune o crear sistemas de retriever robustos** (Knowledge Graphs).

Los fundamentos vistos son agnósticos a las tecnologías/frameworks.

Recuerden mantenerse al día mediante papers, posts, etc.

Gracias por su atención y dedicación.

Recuerden que los grandes retos traen grandes aprendizajes.