Simposio CEA de Robótica, Bioingeniería, Visión por Computador y Automática Marina 2026

LLMs con razonamiento para tareas asistivas en entornos domésticos

Proctor, F.*, Mora, A., Prados, A., Espinoza, G., Moreno, L., Garrido, S.

RoboticsLab, Departamento de Ingeniería de Sistemas y Automática, Universidad Carlos III de Madrid, Av. de la Universidad, 30, 28911 Leganés, España.

Resumen

Los planificadores de tarea robóticos basados en modelos grandes de lenguaje (LLMs) presentan sólidos resultados como planificadores de tarea generalistas. SayPlan combina LLMs pre-entrenados con escenas de grafo tridimensionales para abordar limitaciones en horizontes de planificación y complejidad del entorno. Sin embargo, la implementación del artículo original usa costosas APIs de LLM, y limita los experimentos a entornos extensos pero simples, no aptos para robots asistenciales domésticos que requieren un uso constante con docenas de objetos por habitación. En este artículo, presentamos mejoras al enfoque de SayPlan para evaluar su aptitud en aplicaciones de bajo coste en entornos domésticos complejos. Introducimos una arquitectura modular para evaluar dichas mejoras mediante una serie de estudios de ablación.

Palabras clave: Planificación de tareas y movimiento, Robótica impulsada por IA, Robots móviles autónomos, LLMs para modelado y control, Control eficiente en datos mediante modelos fundacionales.

Reasoning LLMs for Assistive Tasks in Domestic Environments

Abstract

LLM-based robotic task planners present strong results as generalist task planners. SayPlan combines pre-trained LLMs with three-dimensional scene graphs to overcome limitations in planning horizon and environment complexity. However, its paper implementation involves expensive LLM APIs and limits experiments to large but simple environments, not suitable to assistive domestic robots which require consistent use in complex environments with dozens of objects per room. In this paper, we introduce improvements to SayPlan's approach to increase suitability to low-cost, complex-environment domestic applications. We introduce a modular architecture to enable evaluation of such improvements through a series of ablation studies.

Keywords: Task and motion planning, AI-powered robotics, Autonomous mobile robots, LLMs for modeling and control, Data-efficient control via foundation models.

1. Introducción

La planificación de tareas es un elemento clave de un sistema robótico autónomo. A medida que los robots se utilizan cada vez más en entornos humanos no estructurados, la variedad y complejidad de las tareas requeridas aumenta considerablemente. Los entornos domésticos reales presentan un reto particular, caracterizado por una explosión combinatoria de objetos y configuraciones espaciales densamente pobladas (habitualmente más de 30 objetos por habitación). Además de este desafío de escala espacial, los trabajos existentes en esta materia introducen vulnerabilidades críticas en los bucles de control al depender de modelos grandes de lenguaje (*Large Language Models* o LLMs) servidos a través de *Application Programming Interfaces* (APIs) comerciales. Esta dependencia de la nube genera altos costes continuos y riesgos de privacidad de datos. A esto se suma que la mayoría de los enfoques actuales se basan en modelos predictivos tradicionales, también conocidos como modelos de Sistema 1, que no aprovechan los recientes avances en modelos con capacidades intrínsecas de razonamiento, lo que reduce su capacidad de resolver tareas complejas.

En este trabajo partimos de la estructura presentada en SayPlan (Rana et al., 2023), la cual interpreta instrucciones en lenguaje natural ambiguo utilizando grafos de escena tridimensionales (3DSG), proponiendo las siguientes mejoras:

- **Adaptación a entornos domésticos densos:** Optimiza-

*Autor para correspondencia: fproctor@pa.uc3m.es

mos la representación en el 3DSG y la gestión del tamaño de contexto para superar el cuello de botella de memoria que supone planificar con una densidad de objetos casi diez veces superior a la evaluada en el trabajo original.

- **Planificación local (*offline*):** Transicionamos la arquitectura hacia el uso de LLM locales desplegados en hardware convencional (modelos de 7 a 14 mil millones de parámetros), eliminando la dependencia de la nube y adaptando el planificador a las limitaciones de *Video Random-Access Memory* (VRAM).
- **Integración de LLM de razonamiento:** Incorporamos modelos destilados con capacidad de cadena de pensamiento (*Chain-of-Thought* o modelos de Sistema 2) dotando al robot de la capacidad de razonar paso a paso para evitar errores en tareas complejas.

Para validar las ideas propuestas, llevamos a cabo una serie de experimentos de ablación evaluando la tasa de éxito y la ejecutabilidad de los planes en un simulador de grafos de escena 3D basado en entornos domésticos de VirtualHome. Los resultados obtenidos demuestran que la optimización de *prompts* y la integración de modelos de razonamiento en fases divididas mejoran sustancialmente la resolución de tareas de horizonte largo. Concretamente, demostramos que es posible elevar drásticamente el rendimiento de los modelos locales, logrando resultados en hardware de consumo que rivalizan con los de las APIs comerciales de gama alta, validando así la viabilidad de una planificación robótica autónoma, segura y sin conexión.

2. Trabajo relacionado

La planificación automática requiere lenguajes de especificación y algoritmos de resolución. Evaluamos los trabajos relacionados basándonos en su tasa de éxito y en la escala de las tareas (horizonte y complejidad). Los marcos de programación lógica, como ProLog (Wielemaker et al., 2012), utilizan lenguajes como *Planning Domain Definition Language* (PDDL) (Kovacs, 2011) para encontrar soluciones deterministas. Sin embargo, su gran inconveniente es que exigen una definición manual, rígida y exhaustiva de todos los recursos y variables del entorno. Alternativamente, los métodos basados en aprendizaje reducen esta necesidad de conocimiento humano (Garrett et al., 2021), dividiéndose en enfoques puramente de aprendizaje e híbridos.

2.1. Métodos basados puramente en aprendizaje

ProgPrompt (Singh et al., 2023) propone una estrategia de pocos ejemplos (*few-shot*) donde un LLM preentrenado genera planes estructurados en Python basados en objetos y acciones. Sin embargo, aunque es capaz de operar en simulaciones con 115 objetos, obtiene apenas un 48 % de tasa de éxito. Por su parte, el *Hierarchical Planner* de Takayanagi et al. (2019) entrena políticas mediante la codificación de grafos jerárquicos para generalizar tareas nuevas; no obstante, aunque alcanza una alta tasa de éxito (76 %), la escala de los entornos en los que opera es extremadamente limitada, restringiéndose a escenarios simples con apenas 8 objetos (ver Tabla 1). Otro enfoque, Liang et al. (2025), utiliza la Optimización de Preferencia Estructurada (SPO) y aprendizaje por currículo evaluándose en entornos de 55 objetos, pero presenta el fallo de que su tasa de éxito global se estanca en el 48 % y decae drásticamente (al 16 %) cuando se enfrenta a tareas de horizonte largo.

En el ámbito de las representaciones espaciales, los 3DSG han demostrado ser eficaces (Bae et al., 2022). Basándose en ellos, GRID (Ni et al., 2024) emplea redes de atención para predecir subtareas. Sus principales inconvenientes son su moderada tasa de éxito (64 %) limitada a escenas de tamaño medio (70 objetos), la necesidad de un entrenamiento muy costoso con conjuntos de datos masivos (82.000 subtareas) específicos para cada escena, y su incapacidad para manejar de forma realista objetos duplicados. Finalmente, los modelos de Visión-Lenguaje-Acción intentan unificar percepción y acción, pero siguen fuertemente limitados por el débil razonamiento espacial de los LLM y su enfoque exclusivo en tareas de horizonte corto (Li et al., 2025).

2.2. Métodos híbridos

Los métodos híbridos combinan el sentido común de los modelos basados en aprendizaje con resolutores clásicos. ViLaIN (Shirai et al., 2024) traduce imágenes a definiciones PDDL mediante modelos de Visión-Lenguaje y *re-prompting* correctivo; sin embargo, su aplicabilidad se restringe únicamente a entornos pequeños (apenas 10 objetos, como se observa en la Tabla 1). SayPlan (Rana et al., 2023) aborda tareas de horizonte largo realizando búsquedas semánticas sobre 3DSG para reducir el tamaño del grafo antes de aplicar un planificador clásico de trayectorias. Seleccionamos este trabajo como base para la arquitectura propuesta, porque cumple con el criterio principal: ha sido validado en entornos con una gran cantidad de objetos (151 objetos, representativo de la densidad de un entorno doméstico) y, dentro de los enfoques de gran escala, es el que alcanza la mayor tasa de éxito (73 %). Sin embargo, tiene el gran inconveniente de depender de LLMs alojados en la nube con un consumo de *tokens* extremadamente alto, lo que dificulta su viabilidad económica e introduce riesgos de ciberseguridad.

Las secciones siguientes describen la reimplementación de este artículo, siguiendo las sugerencias de los autores, para obtener métricas de rendimiento. A continuación, proponemos mejoras para el rendimiento específico en entornos complejos con hardware *offline* convencional. Finalmente, realizamos estudios de ablación para medir el impacto de estos cambios.

3. Implementación del artículo original

Describimos nuestra implementación de SayPlan (Rana et al., 2023) y su integración en nuestro entorno de simulación. Diseñamos una arquitectura modular que replica la configuración original para evaluar su rendimiento base y, posteriormente, introducir mejoras mediante estudios de ablación. Su flujo de trabajo se estructura en tres fases secuenciales (Figura 1):

3.1. Fase 1: Búsqueda semántica jerárquica

Para evitar exceder la ventana de contexto en entornos a gran escala, el LLM no recibe el grafo completo. En su lugar, inicia con un grafo contraído (nodos de nivel superior) y utiliza llamadas a una API (`expand()` y `contract()`) para explorarlo semánticamente. La extracción de este subgrafo relevante reduce el consumo de *tokens* hasta un 82, 1 %, enfocando al modelo exclusivamente en las entidades útiles para la tarea.

Tabla 1: Comparativa de métodos de planificación basados en aprendizaje, ordenada por tasa de éxito.

Artículo	Entorno de simulación y entrada planificador	Planificación de tareas	Complejidad de entorno	Tasa de éxito
Hierarchichal Planner (LLM)	Gazebo Environment Variables	Hierarchical Task Transformer-based Planner	Blocksworld y Table Arrange, 8 objetos	76 %
SayPlan (Híbrido)	Stanford 3D Scene Graph dataset	Búsqueda Semántica y Planificación LLM y Dijkstra	Oficina, 151 objetos	73 %
ViLaIN (Híbrido)	Modelo V&L y ViLaIN	PDDL (FastDownward)	Blocksworld y Hanoi, 10 objetos	71 %
GRID (LLM)	Unity con Generador 3DSG	GRID	70 objetos	64 %
ProgPrompt (LLM)	VirtualHome con detector de objetos ViLD	ProgPrompt	VirtualHome, 115 objetos	48 %
SPO (LLM)	VirtualHome y Habitat con dataset ExtendaBench	LLM con SPO	VirtualHome, 55 objetos	48 %

3.2. *Fase 2: Integración de planificación clásica*

SayPlan desacopla la semántica de la navegación: el LLM decide los destinos de alto nivel y un planificador clásico (Dijkstra) calcula la ruta exacta. Aunque omitimos experimentalmente esta etapa al delegar la navegación en nuestro simulador de alto nivel, este enfoque acorta el horizonte de planificación del LLM, previniendo la alucinación de rutas inviables y centrando su carga cognitiva en la lógica de manipulación.

3.3. *Fase 3: Replanificación iterativa con retroalimentación*

Dado que los planes iniciales suelen vulnerar restricciones físicas (p. ej., interactuar con objetos en contenedores cerrados), estos se validan previamente. Para ello, dotamos a nuestro simulador 3DSG de un agente virtual que ejecuta lógicamente el plan propuesto. Si detecta un fallo, devuelve una retroalimentación textual que se añade al *prompt* del LLM, forzando una replanificación iterativa corregida.

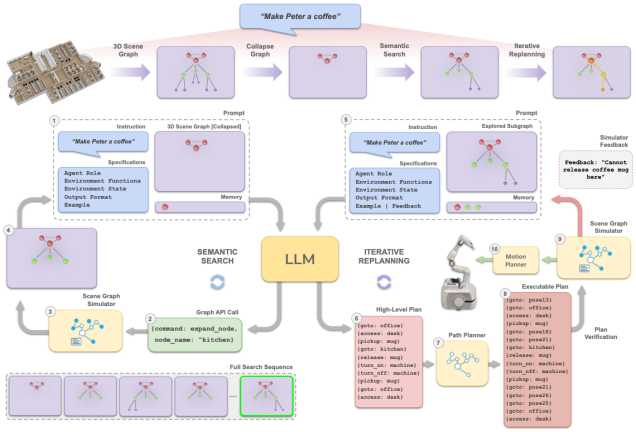


Figura 1: Esquema de la arquitectura SayPlan (Rana et al., 2023).

4. Mejoras propuestas

Como se ha observado anteriormente, el planificador SayPlan original se implementó con restricciones diferentes a las de nuestra aplicación propuesta. Además, se han producido grandes avances en los LLM, tanto en su capacidad de razonamiento como en su eficiencia. Proponemos mejoras para dar respuesta a un entorno más complejo y a una configuración de *Graphical Processing Unit* (GPU) convencional sin conexión (*offline*), detalladas en las siguientes secciones y en la Figura 2.

4.1. *Optimización del prompt de Sistema 1*

Como se aprecia en Rana et al. (2023), los *prompts* utilizados en SayPlan son extensos e incluyen información que puede ser comprimida. Nuestro primer estudio de ablación busca aplicar las mejores prácticas de los LLM de Sistema 1 para reducir el número de *tokens*. Concretamente, llevamos a cabo los siguientes cambios:

1. **Eliminación de información de pose innecesaria:** La implementación original de SayPlan incluye nodos de pose, que se utilizan para conectar habitaciones. Nosotros eliminamos dichos nodos, sustituyéndolos por una arista directa entre habitaciones que representa la conectividad. Este paso nos permite reducir el tamaño del *prompt* en aproximadamente un 10 %, pasando de 2473 a 2226 *tokens*.
2. **Anclaje de contexto (Context Anchoring):** Cuando las instrucciones se colocan únicamente al principio del *prompt*, seguidas de 1.500 *tokens* de ejemplos y un 3DSG, la atención hacia esa regla inicial se diluye. El anclaje de contexto soluciona esto estableciendo la regla al principio y repitiéndola al final del *prompt*. Esto obliga a que los mayores pesos de atención del modelo se dirijan directamente a las restricciones del esquema.

Como se puede observar en la Figura 2 (“*1 - System 1 Prompt Optimisation*”), esta optimización únicamente cambia el *prompt* y se evalúa con un LLM de Sistema 1.

4.2. *Analizador sintáctico LLM (LLM Parser)*

Los LLM suelen tener dificultades para devolver una salida estructurada, especialmente ante una tarea compleja y un contexto extenso. Requerimos que el LLM presente su salida tanto para la búsqueda semántica como para las etapas de planificación en formato *JavaScript Object Notation* (JSON). Para evitar fallos en los que el LLM devuelve una respuesta correcta pero con un formato incorrecto, utilizamos un segundo LLM para verificar y corregir el formato y la sintaxis de la respuesta del primer modelo (véase “*4 - LLM Parsing*” en la Figura 2). Cabe destacar que este segundo LLM actúa como un corrector estricto de JSON, sin tener conocimiento del contexto de la tarea.

4.3. *Prompts de razonamiento de tipo zero-shot*

Proponemos sustituir los LLM de Sistema 1 utilizados en el artículo original por LLM de Sistema 2. La metodología de *prompting* utilizada para un LLM de Sistema 2 difiere de la de un Sistema 1. En Guo et al. (2025), DeepSeek-AI menciona explícitamente que el *prompting* de pocos ejemplos (*few-shot*) degradaba el rendimiento del modelo en tareas de razonamiento en comparación con el de cero ejemplos (*zero-shot*), ya que el modelo intenta replicar la lógica de los ejemplos en lugar de la suya propia. En Cheng et al. (2025), los autores realizan experimentos para demostrar que los ejemplos de CoT no conducen a mejores resultados, ya que los modelos solo utilizan estos ejemplos para calibrar el formato de salida.

Por lo tanto, actualizamos los *prompts* para que se basen en un enfoque *zero-shot*. No mostramos ejemplos de planes JSON anteriores. En su lugar, enunciamos las reglas brutas y el modelo utiliza su cadena de pensamiento interna para generar planes dinámicamente, verificando su propio trabajo paso a paso antes de emitir la acción de búsqueda de grafo o el plan final (véase "*2 - System 2 Prompting*" en la Figura 2).

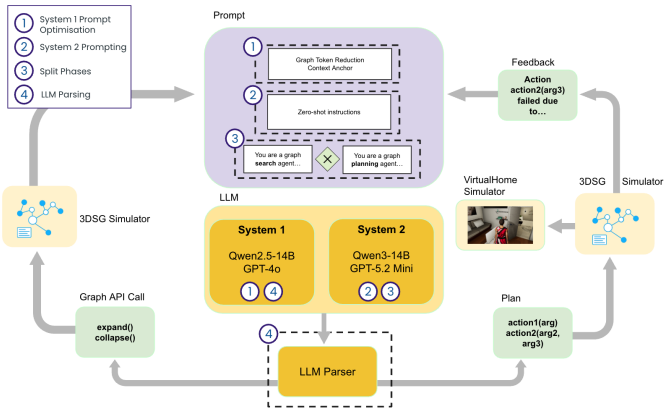


Figura 2: Implementación de mejoras propuestas a la arquitectura de SayPlan. Cada una de las cuatro mejoras puede ser activada independientemente para realizar estudios de ablación.

4.4. División de fases

Como se puede observar en la Figura 1, el LLM actúa tanto como agente de búsqueda en grafos como planificador de tareas. Estas tareas se realizan de forma secuencial. El LLM primero debe encontrar un subgrafo relevante para la instrucción y solo entonces debe intentar planificar. Sin embargo, la estructura de *prompting* utilizada en Rana et al. (2023) implica describir y proporcionar ejemplos *few-shot* tanto de las etapas de exploración como de las de planificación de principio a fin. Esto significa que durante una solicitud, aunque el modelo solo necesita realizar una de las dos tareas, se le somete a la carga cognitiva inicial de ambas, teniendo que procesar ejemplos y lógica para las dos simultáneamente.

Proponemos dividir estrictamente estas dos fases y los *prompts* entregados al modelo, dependiendo de si está realizando una búsqueda o una planificación (véase Figura 2, *prompt* 3). Dividimos la forma en que interactuamos con el LLM, manteniendo la replanificación iterativa y el enfoque *zero-shot*:

- **Fase de búsqueda en el grafo:** Aquí, el modelo recibe instrucciones para actuar como un agente de búsqueda en grafos, con la única instrucción de encontrar iterativamente un subgrafo relevante para la instrucción. No tiene conocimiento de que más adelante estará planificando una tarea y, por lo tanto, puede centrarse únicamente en la tarea de explorar el grafo.

- **Fase de planificación:** Aquí, el modelo recibe instrucciones para generar un plan basado en un subgrafo. No sabe que el grafo que se le proporciona es un subgrafo, y su enfoque se centra enteramente en generar un plan factible dado ese grafo.

Este enfoque busca mejorar notablemente el rendimiento de los LLM locales, ya que su menor cantidad de parámetros los hace vulnerables a la saturación del contexto, induciendo sesgos de recencia y la integración errónea del *prompt*.

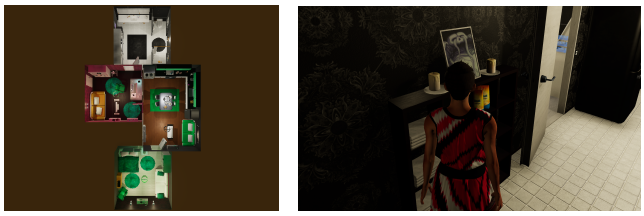
5. Configuración experimental

5.1. Simulación física y representación 3DSG

El enfoque de nuestro estudio es evaluar la capacidad del planificador de alto nivel para generar planes correctos y ejecutables. Gracias a la modularidad de la arquitectura, el planificador no necesita interactuar directamente con la simulación física, sino que razona y valida la viabilidad de los planes a través de un simulador de 3DSG. Este simulador de 3DSG es un elemento crítico de la arquitectura SayPlan y lo hemos implementado utilizando la estructura `MultiDiGraph` de la librería NetworkX (Hagberg et al., 2008). En este espacio abstracto, la búsqueda semántica se logra mediante llamadas a la API para contraer o expandir partes del grafo, mientras que la validación del plan y la replanificación iterativa se consiguen simulando las acciones de forma lógica sobre los nodos y aristas del 3DSG.

Para respaldar este modelo abstracto con una simulación física, los requisitos de nuestro planificador exigían un entorno doméstico realista con decenas de objetos por habitación, que permitiera extraer los 3DSG y ejecutar tanto acciones primitivas como planificación de trayectorias. Tras revisar la literatura, decidimos utilizar VirtualHome (Puig et al., 2018), un simulador ampliamente adoptado que facilita la reutilización de conjuntos de datos y proporciona de forma nativa grafos de entorno (*Environment Graphs*).

Conectamos VirtualHome a nuestro simulador de 3DSG (NetworkX) mediante la API de VirtualHome y un módulo de generación propio. De este modo, el flujo de trabajo es completo: una vez que el planificador genera un plan y este es validado lógicamente por el simulador de 3DSG, lo transferimos y ejecutamos en la simulación física de VirtualHome en bucle abierto (*open-loop*), como se observa en la Figura 3.



(a) (b)

Figura 3: Simulador VirtualHome. (a) Una de las ocho escenas utilizadas. (b) Agente ejecutando un plan.

5.2. Selección de LLM y Hardware

Nuestro enfoque aprovecha los recientes avances en modelos de lenguaje (LLM) para permitir una planificación robótica viable en entornos domésticos. Para evaluar nuestras mejoras de forma exhaustiva, la configuración experimental se fundamenta en los siguientes cuatro pilares:

- **Hardware representativo (*Edge Computing*):** Evaluamos la viabilidad local (*offline*) utilizando una NVIDIA GeForce RTX 2080 Ti (11 GB de VRAM), estableciendo un techo de memoria realista que equilibra la carga gráfica y la inferencia en hardware de consumo.

- **LLMs locales *offline*:** Desplegamos modelos compactos (14B de parámetros) en local mediante Ollama. Destaca Qwen3-14B, que compensa su tamaño mediante destilación cognitiva basada en cientos de miles de trayectorias *Chain-of-Thought* de modelos maestros, aprendiendo a deducir, planificar y autocorregirse de forma autónoma.
- **LLMs API *online*:** Empleamos modelos de trillones de parámetros sin restricciones computacionales disponibles a través de la API de OpenAI. GPT-4o actúa como Sistema 1, mientras que GPT-5.2 Mini actúa como Sistema 2 para evaluar el impacto del procesamiento lógico de múltiples pasos.

El resumen de los modelos seleccionados para los experimentos de ablación se detalla en la Tabla 2.

Tabla 2: Resumen de los LLM seleccionados para los estudios de ablación.

Modelo	Sistema	Despliegue	Características Principales
GPT-4o	1	API *online*	Reemplazo eficiente de GPT-4 (utilizado en artículo original) manteniendo métricas de éxito.
Qwen2.5-14B	1	Local	Alta capacidad cognitiva en formato compacto.
GPT-5.2 Mini	2	API *online*	Razonamiento CoT de última generación sin restricciones de recursos.
Qwen3-14B	2	Local	Destilación cognitiva de modelos maestros. Alta capacidad de razonamiento CoT.

5.3. Instrucciones, planes y entornos

Para evaluar las distintas variantes del planificador, empleamos un conjunto de 16 pares de instrucciones y 3DSGs, acompañados de sus respectivos planes de referencia (*ground truth*) generados manualmente. Estas pruebas consisten en tareas domésticas cotidianas construidas a partir de las mismas acciones primitivas definidas en SayPlan. Las 16 instrucciones se distribuyen a través de 8 escenas virtuales con distintos niveles de complejidad espacial. Concretamente, la mitad de estas escenas se catalogan como "pequeñas" (con apenas 4 objetos o activos por habitación), mientras que las 4 restantes se clasifican como "grandes" (con una densidad aproximada de 30 elementos por estancia). Asimismo, el conjunto de datos equilibra la exigencia temporal: comprende 8 tareas de horizonte corto (*short-horizon*) y 8 de horizonte largo (*long-horizon*), definidas en función de si el plan óptimo requiere menos o más de 8 pasos, respectivamente, para completarse.

6. Resultados

Evaluamos las variantes del planificador frente al simulador 3DSG (garantizando su portabilidad a VirtualHome) utilizando dos métricas principales:

Tasa de éxito (SR): Esta es una métrica binaria de extremo a extremo que representa la medida definitiva del rendimiento del planificador. Una tarea se considera exitosa si el plan es ejecutable, respetando tanto la conectividad jerárquica espacial como las restricciones físicas de sentido común, y el estado final del simulador de 3DSG coincide con las condiciones del objetivo especificadas por la instrucción en lenguaje natural; por ejemplo, verificando que un objeto específico esté contenido dentro de un activo de destino.

Número de replanificaciones: Registramos el número de pasos de replanificación iterativa necesarios para lograr un plan ejecutable, midiendo la eficiencia del proceso de razonamiento y la eficacia de la retroalimentación textual proporcionada por el simulador. En consonancia con Rana et al. (2023), imponemos un límite de cinco iteraciones de replanificación por tarea.

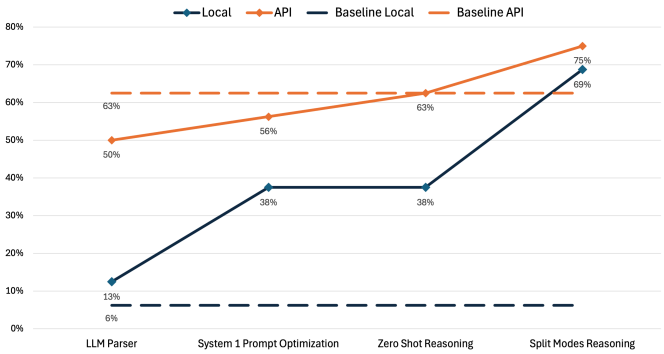


Figura 4: Tasas de éxito base (líneas discontinuas) y para cada fase de ablación, evaluadas con los modelos en línea y locales.

Las tasas de éxito de los experimentos de ablación pueden verse en la Figura 4 y en la Tabla 3. Tanto para los experimentos con LLM en la nube (API) como con los alojados localmente, comenzamos obteniendo un rendimiento de referencia en nuestro conjunto de datos de instrucciones y planes reales. Existe una gran diferencia de rendimiento inicial entre las API y los LLM locales: mientras las API alcanzan una SR inicial del 63 %, los modelos locales solo logran un 6 % (líneas discontinuas en la Figura 4). El modelo pequeño, de 14 mil millones de parámetros, tiene dificultades con la carga de seguir simultáneamente las instrucciones de planificación y de formato, fallando el 50 % de las veces debido a errores de formato en la salida. Del resto, solo una tarea se completa con éxito.

6.1. Mejoras con la API

Las mejoras propuestas presentan ganancias marginales sobre la implementación original del artículo cuando se aplican a LLM de gama alta vía API. Como se observa en la Figura 4, el *parsing* por LLM y la optimización del *prompt* con modelos de Sistema 1 solo degradan el rendimiento del planificador en comparación con la línea de base. Solo se produce una mejora al utilizar un modelo de razonamiento y aplicar la división de las fases de búsqueda y planificación, lo que conduce a un aumento del rendimiento del 63 % al 75 % de SR.

6.2. Mejoras en local

El análisis sintáctico por LLM (LLM Parser) muestra algunas mejoras en el rendimiento de los modelos de Sistema 1 ejecutados localmente, elevando la SR del 6 % al 13 %, debido a la corrección de errores de formato producidos por la sobrecarga del modelo local. Optimizando los *prompts* de Sistema 1 se alcanzan avances sólidos en la SR, pasando del 6 % inicial al 38 %. Esto pone de manifiesto la importancia de reducir los *tokens* al máximo, así como de anclar el contexto al final del *prompt*, reduciendo el sesgo de recencia.

Planteamos que los modelos de razonamiento pueden permitir grandes saltos de rendimiento en hardware local. El uso de

Tabla 3: Tasas de éxito (SR) y duración y número medio de replanificaciones para tareas exitosas

	Experimento	Media	Entorno Pequeño	Entorno Grande	Tarea Corta	Tarea Compleja	Replanificaciones	Duración
Local	Baseline	6 %	13 %	0 %	0 %	11 %	3.00	50s
	LLM Parser	13 %	25 %	0 %	29 %	0 %	1.50	1m42s
	System 1 P.O.	38 %	50 %	25 %	57 %	22 %	0.66	**33s**
	Zero-Shot Reasoning	38 %	38 %	38 %	57 %	22 %	1.00	3m02s
	Split Modes Reasoning	**69 %**	**75 %**	**63 %**	**86 %**	**56 %**	**0.55**	2m22s
API	Baseline	63 %	75 %	**50 %**	**86 %**	44 %	0.30	44s
	LLM Parser	50 %	75 %	25 %	43 %	56 %	0.38	38s
	System 1 P.O.	56 %	75 %	38 %	71 %	44 %	0.89	**19s**
	Zero-Shot Reasoning	63 %	88 %	38 %	71 %	56 %	0.80	1m07s
	Split Modes Reasoning	**75 %**	**100 %**	**50 %**	71 %	**78 %**	**0.08**	1m

un modelo de razonamiento con *prompting zero-shot* genera un salto del 6 % al 38 % de SR. Sin embargo, el mayor incremento de rendimiento proviene de ejecutar el planificador en modos divididos. Esto le permite centrarse únicamente en las tareas de búsqueda en el grafo o de planificación, aprovechando plenamente sus capacidades de razonamiento. Alcanza una SR del 69 %, un aumento notable respecto al 6 % de la implementación original ejecutada en local, y un rendimiento casi comparable al mejor resultado de la API (75 %).

7. Conclusiones y Trabajo Futuro

Los resultados demuestran que la brecha de rendimiento entre la nube y el entorno local se cierra mediante el uso de modelos de Sistema 2 y la modularización de tareas. Mientras que un modelo local de 14B falla inicialmente debido a la sobrecarga de formato y lógica (6 % SR), la transición a un modelo de razonamiento con una estructura de fases divididas permite alcanzar una tasa de éxito del 69 %. Este salto reduce la dependencia de las API en la nube (75 % SR), demostrando que la destilación *CoT* en hardware convencional es suficiente para una planificación robótica autónoma, precisa y ejecutable.

Nuestro estudio abre tres vías principales para optimizar la planificación robótica autónoma. En primer lugar, explorar LLMs que fuercen la generación mediante un *JSON Schema* estricto y aplicar técnicas de compresión como *Token-Oriented Object Notation* (TOON) mitigaría los errores de formato y la carga de *tokens*. En segundo lugar, investigar el enrutamiento dinámico de modelos permitiría evaluar la complejidad de la tarea en tiempo real, asignando escenarios sencillos a modelos reactivos (Sistema 1) y complejos a sistemas deliberativos (Sistema 2), complementado con ajustes heurísticos de la temperatura para fomentar la exploración durante la replanificación. Finalmente, de cara a la autonomía física, es imperativo trasladar estos enfoques más allá de la simulación, sustituyendo el 3DSG estático por generadores de grafos en tiempo real como Hydra (Hughes et al., 2022), para validar la resiliencia del planificador ante el ruido y la incertidumbre del mundo real.

Agradecimientos

Este trabajo ha sido apoyado por el proyecto Advanced Mobile dual-arm Manipulator for Elderly People Attendance (AMME) (PID2022-139227OB-I00), financiado por el Ministerio de Ciencia e Innovación.

Referencias

Bae, J., Shin, D., Ko, K., Lee, J., Kim, U.-H., 2022. A survey on 3d scene graphs: Definition, generation and application. In: International Conference on Robot Intelligence Technology and Applications. Springer, pp. 136–147.

Cheng, X., Pan, C., Zhao, M., Li, D., Liu, F., Zhang, X., Zhang, X., Liu, Y., 2025. Revisiting chain-of-thought prompting: Zero-shot can be stronger than few-shot. arXiv preprint arXiv:2506.14641.

Garrett, C. R., Chitnis, R., Holladay, R., Kim, B., Silver, T., Kaelbling, L. P., Lozano-Pérez, T., 2021. Integrated task and motion planning. Annual review of control, robotics, and autonomous systems 4 (1), 265–293.

Guo, D., Yang, D., Zhang, H., Song, J., Wang, P., Zhu, Q., Xu, R., Zhang, R., Ma, S., Bi, X., et al., 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. arXiv preprint arXiv:2501.12948.

Hagberg, A. A., Schult, D. A., Swart, P. J., Aug 2008. Exploring network structure, dynamics, and function using networkx. In: Varoquaux, G., Vaught, T., Millman, J. (Eds.), Proceedings of the 7th Python in Science Conference (SciPy2008). Pasadena, CA USA, pp. 11–15.

Hughes, N., Chang, Y., Carlone, L., 2022. Hydra: A real-time spatial perception system for 3d scene graph construction and optimization. arXiv preprint arXiv:2201.13360.

Kovacs, D. L., 2011. Bnf definition of pddl 3.1. Unpublished manuscript from the IPC-2011 website 15.

Li, D., Zhang, Y., Cao, M., Liu, D., Xie, W., Hui, T., Lin, L., Xie, Z., Li, Y., 2025. Towards long-horizon vision-language-action system: Reasoning, acting and memory. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 6839–6848.

Liang, X., Lin, M., Ruan, W., Xu, R., Liu, Y., Chen, J., Lin, B., Zhuang, Y., Liang, X., 2025. Structured preference optimization for vision-language long-horizon task planning. arXiv preprint arXiv:2502.20742.

Ni, Z., Deng, X., Tai, C., Zhu, X., Xie, Q., Huang, W., Wu, X., Zeng, L., 2024. Grid: Scene-graph-based instruction-driven robotic task planning. In: 2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, pp. 13765–13772.

Puig, X., Ra, K., Boben, M., Li, J., Wang, T., Fidler, S., Torralba, A., 2018. Virtualhome: Simulating household activities via programs. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 8494–8502.

Rana, K., Haviland, J., Garg, S., Abou-Chakra, J., Reid, I., Suenderhauf, N., 2023. Sayplan: Grounding large language models using 3d scene graphs for scalable robot task planning. In: Conference on Robot Learning. PMLR, pp. 23–72.

Shirai, K., Beltran-Hernandez, C. C., Hamaya, M., Hashimoto, A., Tanaka, S., Kawaharazuka, K., Tanaka, K., Ushiku, Y., Mori, S., 2024. Vision-language interpreter for robot task planning. In: 2024 IEEE International Conference on Robotics and Automation (ICRA). IEEE, pp. 2051–2058.

Singh, I., Blukis, V., Mousavian, A., Goyal, A., Xu, D., Tremblay, J., Fox, D., Thomason, J., Garg, A., 2023. Progprompt: program generation for situated robot task planning using large language models. Autonomous Robots 47 (8), 999–1012.

Takayanagi, T., Kurose, Y., Harada, T., 2019. Hierarchical task planning from object goal state for human-assist robot. In: 2019 IEEE 15th International Conference on Automation Science and Engineering (CASE). IEEE, pp. 1359–1366.

Wielemaker, J., Schrijvers, T., Triska, M., Lager, T., 2012. Swi-prolog. Theory and Practice of Logic Programming 12 (1-2), 67–96.