Agentes basados en LLMs para la derivación de tareas de desarrollo backend a partir de requerimientos textuales

Nicolás Miccio Palermo[1]

[1]ISISTAN, CONICET-UNICEN – Tandil, Buenos Aires, Argentina

nicolas.miccio@isistan.unicen.edu.ar

Abstract. *Backend development faces challenges in transforming informal functional requirements into precise and implementable tasks. This work explores the use of Large Language Models (LLMs) as collaborative, task oriented agents to support early stages of the software development lifecycle. We propose a modular multi-agent workflow in which specialized agents refine requirements, decompose functionality into services, define API contracts, specify test suites, and ensure consistency. The goal is to assist developer by reducing manual effort, coordination overhead, and misalignment, while improving speed, and quality of backend development.*

Resumen. *El desarrollo backend presenta desafíos para transformar requisitos funcionales informales en tareas precisas e implementables. Este trabajo explora el uso de Modelos de Lenguaje de Gran Escala (LLMs) como agentes colaborativos y orientados a tareas para apoyar las etapas tempranas del ciclo de vida del software. Se propone un flujo de trabajo modular multi-agente en el que un conjunto de agentes especializados asisten a un desarrollador a refinar requisitos, descomponer funcionalidades en servicios, definir contratos de API, especificar suites de pruebas y asegurar consistencia. El objetivo es reducir el esfuerzo manual, la sobrecarga de coordinación y posibles desalineamientos, mejorando la velocidad y calidad del desarrollo.*

1. Introducción y contexto

El desarrollo de sistemas *backend* confiables continúa siendo un desafío central dentro del ciclo de vida del software, especialmente en proyectos caracterizados por requerimientos cambiantes, equipos distribuidos y alta complejidad técnica. Una de las principales dificultades radica en la traducción de especificaciones funcionales textuales, frecuentemente informales y ambiguas, en tareas de desarrollo precisas y accionables. Este proceso implica identificar funcionalidades clave, asignarlas a componentes del sistema, y definir de manera consistente las interacciones y restricciones entre ellos. En este contexto, la comunidad de arquitectura de software ha enfatizado la importancia de documentar explícitamente las decisiones clave de diseño [Capilla et al. 2016], con el fin de mejorar la trazabilidad, la comunicación entre los actores involucrados y la sostenibilidad de los proyectos a lo largo del tiempo.

El diseño de un sistema no sólo debe garantizar el cumplimiento de los requerimientos funcionales definidos por los distintos *stakeholders*, sino además satisfacer un conjunto de atributos de calidad, también llamados requerimientos no funcionales

[Bachmann et al. 2005]. Estos atributos pueden estar sujetos a conflictos entre ellos y por lo tanto no siempre pueden ser satisfechos simultáneamente [Bass et al. 2021].

En la práctica, esta traducción desde requerimientos textuales hacia artefactos de diseño estructurados continúa siendo un proceso mayormente manual y altamente dependiente de la experiencia de analistas y arquitectos [Ferrari and Madhavji 2008]. La descomposición funcional en servicios, la definición de APIs y la identificación de responsabilidades entre componentes suelen involucrar múltiples iteraciones y una intensa coordinación entre distintos roles. Esta situación se vuelve especialmente compleja en proyectos con equipos distribuidos o con requerimientos en constante evolución, donde los costos de comunicación y las decisiones inconsistentes pueden derivar en soluciones de baja calidad.

En los últimos años, la aparición de agentes de software impulsados por Modelos de Lenguaje de Gran Escala (LLMs), entendidos como entidades capaces de planificar, razonar y ejecutar acciones en función de un objetivo dado, ha abierto nuevas oportunidades para asistir distintas actividades del ciclo de desarrollo de software [Wang et al. 2024] [Sami et al. 2024]. No obstante, la mayoría de las propuestas existentes se concentran en las etapas finales, tales como la generación de código, la asistencia durante la implementación o la resolución de *bugs* (por ejemplo, SWE-Bench [Jimenez et al. 2024]). Como consecuencia, el soporte brindado a las fases tempranas de especificación y diseño *backend* continúa siendo limitado, a pesar de que en estas etapas se toman decisiones que condicionan fuertemente la estructura, calidad y evolución del sistema.

Si bien algunos trabajos recientes han explorado el uso de LLMs para analizar requerimientos, proponer estilos arquitectónicos o componer modelos de diseño simples [White et al. 2024], estos enfoques suelen operar de manera puntual o asistida, sin ofrecer mecanismos sistemáticos para coordinar múltiples actividades de diseño ni garantizar trazabilidad entre requerimientos, decisiones y artefactos resultantes. En particular, aún persisten desafíos relacionados con el nivel de autonomía de los agentes, la integración explícita de conocimiento arquitectónico y la capacidad de adaptar el razonamiento a contextos de desarrollo diversos, lo que limita su adopción como asistentes robustos para arquitectos y diseñadores de software [Díaz-Pace et al. 2024].

En este contexto, surge la necesidad de investigar cómo los LLMs, organizados como agentes colaborativos y orientados a tareas específicas, pueden asistir a la transformación de requerimientos funcionales textuales en definiciones estructuradas de componentes *backend*, servicios y tareas de desarrollo. Si bien el enfoque propuesto se centra mayormente en requisitos funcionales, se reconoce que los atributos de calidad y otros requisitos no funcionales influyen de manera significativa en las decisiones de diseño y constituyen un aspecto relevante a considerar durante el desarrollo del trabajo. Abordar este problema permitiría reducir el esfuerzo manual y mejorar la alineación entre requerimientos y diseño. Este trabajo se inscribe en esta problemática y propone un enfoque basado en agentes impulsados por LLMs para apoyar las etapas tempranas del diseño *backend*, con especial énfasis en la derivación de tareas de desarrollo a partir de requerimientos textuales y en la comprensión de las decisiones involucradas durante dicho proceso.

2. Preguntas y objetivos del trabajo

2.1. Preguntas de investigación

Las siguientes preguntas buscan analizar si el enfoque propuesto es técnicamente viable y que impacto real tiene sobre el proceso desarrollo de software.

- RQ1: ¿Cómo pueden agentes basados en LLMs transformar requerimientos funcionales textuales, potencialmente incompletos o ambiguos, en definiciones coherentes y consistentes de componentes *backend* y servicios orientados a APIs?
- RQ2: ¿Qué fortalezas y limitaciones presenta la colaboración entre múltiples agentes especializados, cada uno orientado a distintas actividades del ciclo de vida del software, para la derivación de tareas de desarrollo *backend*?
- RQ3: ¿Cómo impacta el uso de agentes basados en LLMs en términos de esfuerzo, calidad de los artefactos generados y costos de coordinación, en comparación con enfoques tradicionales de diseño y descomposición funcional?

2.2. Objetivos

Con el fin de responder a las preguntas planteadas, el objetivo general de este trabajo es *explorar y evaluar el uso de agentes colaborativos basados en LLMs para apoyar la transformación de requerimientos funcionales textuales en tareas estructuradas de desarrollo backend*. A partir de este objetivo general, se definen los siguientes objetivos específicos.

- Diseñar y desarrollar un enfoque basado en agentes LLMs capaz de interpretar requerimientos funcionales expresados en lenguaje natural y transformarlos en definiciones coherentes de componentes, servicios y contratos *backend*.
- Analizar el comportamiento de un flujo de trabajo multi-agente, identificando las fortalezas y limitaciones de la colaboración entre agentes especializados en distintas actividades del ciclo de vida del software, tales como el refinamiento de especificaciones, el modelado de servicios y la generación de artefactos de diseño.
- Caracterizar el comportamiento del enfoque propuesto en distintos escenarios de requerimientos, analizando cómo diferentes configuraciones de agentes, modelos y fuentes de conocimiento influyen en la coherencia, trazabilidad y características de los artefactos generados.

3. Trabajos relacionados

Los avances recientes en LLMs han impulsado una creciente línea de investigación orientada a su uso como asistentes inteligentes en distintas actividades del desarrollo de software. En particular, numerosos trabajos han explorado el potencial de los LLMs para apoyar tareas de diseño, razonamiento arquitectónico y documentación, aprovechando su capacidad para procesar descripciones textuales expresadas en lenguaje natural.

Dentro de esta línea, algunos estudios han investigado el uso de LLMs para generar y documentar decisiones de diseño arquitectónico, por ejemplo mediante la producción automática de *Architectural Decision Records* (ADRs). Por ejemplo, en [Dhar et al. 2024] analizaron la capacidad de distintos LLMs para capturar decisiones arquitectónicas relevantes a partir de descripciones textuales. De manera complementaria, otros trabajos han estudiado el rol de la IA generativa como soporte para el razonamiento arquitectónico,

ya sea guiando decisiones de diseño [Ozkaya 2024] o imitando actividades típicas del arquitecto humano en contextos colaborativos, tales como análisis, síntesis y evaluación de alternativas de diseño [Ahmad et al. 2023]. Estos enfoques evidencian el potencial de los LLMs para asistir tareas cognitivas complejas, pero se centran principalmente en la generación de explicaciones o decisiones a nivel conceptual.

En esta misma dirección, ArchMind [Díaz-Pace et al. 2024] propuso un asistente basado en agentes basados en LLMs para apoyar la toma de decisiones arquitectónicas. El enfoque recibe como entrada los requerimientos expresados en lenguaje natural junto con una descripción básica del sistema, y genera como salida un conjunto de decisiones de diseño candidatas, acompañadas de explicaciones textuales y documentadas en formato de ADRs. ArchMind pone un fuerte énfasis en la interacción *human-in-the-loop*, delegando en el diseñador el control sobre cuándo y cómo se utilizan las capacidades del asistente. Si bien este enfoque demuestra la utilidad de los LLMs para apoyar el razonamiento arquitectónico y la comprensión de *trade-offs* entre atributos de calidad, su alcance se limita a la asistencia interactiva, los agentes no operan de forma autónoma ni coordinada a lo largo de un flujo de trabajo completo, y el enfoque no aborda explícitamente la derivación de artefactos cercanos a la implementación, como servicios, contratos de APIs o tareas de desarrollo *backend*.

Un paso adicional hacia una mayor autonomía es ReArch [Diaz-Pace et al. 2025], que combinó LLMs con técnicas de planificación, reflexión y búsqueda, apoyándose en *frameworks* como ReAct [Yao et al. 2022] y LATS [Zhou et al. 2023]. A partir de requerimientos textuales y una descripción básica del sistema, ReArch genera y analiza múltiples alternativas de diseño en paralelo, priorizando decisiones candidatas y proporcionando explicaciones que facilitan su comparación. No obstante, al igual que ArchMind, el enfoque se mantiene en el nivel de patrones y decisiones arquitectónicas, sin pretender derivar sistemáticamente artefactos orientados a la implementación. Su objetivo principal es apoyar la generación y discusión de decisiones, más que estructurar el proceso completo de transformación desde requerimientos hacia tareas de desarrollo.

En conjunto, estos trabajos reflejan un interés creciente por utilizar LLMs y enfoques basados en agentes para asistir actividades tempranas del diseño de software, particularmente en lo que respecta al razonamiento arquitectónico y la documentación de decisiones. Sin embargo, aún existen desafíos abiertos relacionados con la coordinación efectiva entre agentes especializados, la integración explícita de conocimiento arquitectónico y la trazabilidad sistemática entre requerimientos textuales, decisiones de diseño y artefactos resultantes. En particular, la transformación estructurada de requerimientos funcionales en definiciones de componentes *backend*, servicios y tareas de desarrollo continúa siendo un aspecto poco explorado.

En este contexto, el enfoque propuesto en este trabajo se diferencia de las propuestas existentes tanto por el tipo de problema abordado como por la forma en que se organiza el razonamiento asistido por LLMs. Mientras que gran parte de los trabajos previos se concentran en asistir etapas específicas del ciclo de vida del software (frecuentemente la implementación o la generación de código), este trabajo pone el foco en las etapas tempranas del diseño *backend*, donde los requerimientos funcionales textuales son transformados en decisiones estructurales y tareas de desarrollo que condicionan de manera significativa la evolución del sistema.

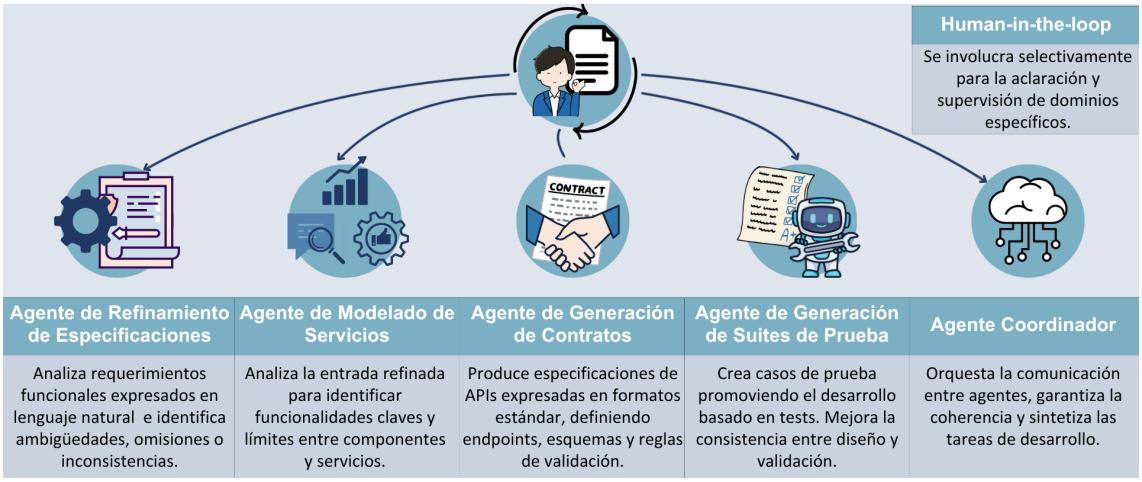

Figura 1. *Workflow* modular basado en múltiples agentes impulsados por LLMs

Finalmente, el trabajo contribuye a una mejor comprensión del *rol del conocimiento arquitectónico explícito* en sistemas basados en LLMs, explorando cómo distintas formas de integración de dicho conocimiento influyen en el comportamiento de los agentes y en la coherencia de los artefactos generados. En conjunto, estos aportes posicionan al trabajo como una propuesta que complementa los enfoques existentes, avanzando desde asistentes puntuales hacia una visión más estructurada del uso de LLMs y agentes en las etapas tempranas del diseño de software.

4. Metodología y solución propuesta

El trabajo se desarrolla siguiendo una metodología *hipotético-deductiva y experimental*, orientada al planteo de problemas, la definición de técnicas para abordarlos y su verificación empírica mediante estudios exploratorios. Las actividades se organizan de manera *iterativa e incremental*, combinando relevamiento bibliográfico, diseño e implementación de prototipos, experimentación y análisis de resultados. En este marco, el trabajo adopta un enfoque exploratorio y progresivo, apoyado en el diseño y construcción de prototipos y en su análisis empírico en distintos escenarios , lo que permite refinar progresivamente las técnicas propuestas a partir del *feedback* obtenido en cada ciclo.

Como objeto central de estudio, se propone una solución basada en un *workflow* modular compuesto por múltiples agentes basados en LLMs (Figura 1), orientado a la transformación semiautomática de requerimientos funcionales de alto nivel en artefactos estructurados de diseño *backend* y tareas de desarrollo.

Un aporte central del trabajo consiste en la *organización del razonamiento asistido por LLMs mediante agentes especializados y colaborativos*, cada uno alineado con responsabilidades tradicionales del diseño *backend*, como el refinamiento de especificaciones, la descomposición en servicios o la definición de contratos de APIs. En este sentido, el enfoque no contempla la generación automática de código fuente, sino la producción de especificaciones y artefactos que guían la implementación posterior. Asimismo, el sistema es agnóstico al LLM utilizado, ya que no depende de capacidades o características específicas de uno en particular, sino de su habilidad general para comprender y generar lenguaje natural, lo que permite adaptar la solución a distintos modelos y a su evolución en el tiempo.

El *workflow* se inicia con un *Agente de Refinamiento de Especificaciones*, cuyo objetivo es analizar requerimientos funcionales expresados en lenguaje natural (como *user histories*) e identificar ambigüedades, omisiones o inconsistencias, produciendo una versión refinada y más precisa de los mismos. A partir de estas especificaciones, el *Agente de Modelado de Servicios* identifica funcionalidades clave, propone límites entre componentes y servicios, y define entidades y responsabilidades principales, generando una descomposición modular del sistema que sirve como base para las decisiones posteriores de diseño *backend*.

Sobre la base de este modelo, el *Agente de Generación de Contratos* traduce la descomposición propuesta en definiciones formales de interfaces, tales como especificaciones de APIs expresadas en formatos estándar, incluyendo *endpoints*, esquemas de datos y reglas de validación. De manera complementaria, el *Agente de Generación de Suites de Prueba* define casos de prueba alineados con los contratos, promoviendo enfoques de desarrollo guiados por *tests* y permitiendo evaluar tempranamente la corrección y completitud de los servicios propuestos.

Finalmente, el *Agente Coordinador* supervisa el *workflow* completo, gestionando dependencias entre agentes y asegurando la coherencia global de los artefactos generados, consolidando los resultados en un conjunto de tareas de desarrollo *backend*. Este agente actúa como un punto de integración, facilitando la trazabilidad entre requerimientos, decisiones de diseño y tareas resultantes.

4.1. Estado actual y avances preliminares

El trabajo se encuentra actualmente en una etapa intermedia del doctorado. Hasta el momento, el foco principal ha estado puesto en la exploración y caracterización del uso de agentes basados en LLMs para apoyar tareas puntuales relacionadas con la descomposición de requerimientos y el razonamiento arquitectónico temprano.

En una primera etapa, se realizó un relevamiento de la literatura y de herramientas existentes relacionadas con LLMs, enfoques basados en agentes y técnicas de recuperación de información, con el objetivo de identificar prácticas habituales, limitaciones recurrentes y oportunidades de mejora en el soporte a etapas tempranas del diseño de software. A partir de este análisis, se desarrollaron prototipos experimentales de asistentes basados en LLMs orientados a la descomposición de requerimientos funcionales y a la identificación de decisiones arquitectónicas relevantes.

Sobre estos prototipos se llevaron a cabo estudios exploratorios utilizando distintos modelos de LLM y diversas configuraciones de provisión de conocimiento, incluyendo enfoques de *Retrieval-Augmented Generation* (RAG) y fuentes de conocimiento arquitectónico explícito, tales como descripciones de patrones de diseño. Estos experimentos permitieron analizar cómo distintas configuraciones influyen en la coherencia de las propuestas generadas, la consistencia entre artefactos y la capacidad de los agentes para explicitar decisiones de diseño. Por ejemplo, se experimentó con diferentes fuentes de conocimiento estructuradas en tres conjuntos diferenciados, donde el asistente fue inicializado alternativamente con cada uno ellos de forma individual, con la combinación de todos, y sin ninguno (configuración *zero-shot*). Las diferencias observadas entre configuraciones se manifestaron principalmente en los patrones sugeridos por el asistente y en las justificaciones asociadas a dichas sugerencias, dado que el formato de salida se man-

tuvo estructuralmente constante. En particular, se identificó que la provisión de distintos conjuntos de patrones conduce a recomendaciones diferentes para un mismo requerimiento, aunque esta variación no siempre muestra una correspondencia directa y clara con el contenido específico de las fuentes suministradas. En algunos casos, el asistente parece comportarse de manera similar a una configuración zero-shot cuando no identifica información relevante en las fuentes provistas, e incluso puede omitir recomendaciones.

Asimismo, se exploraron mecanismos de evaluación automática basados en *LLM-as-a-judge*, con el fin de comparar de manera preliminar el comportamiento de diferentes configuraciones del asistente en escenarios de requerimientos variados. Los resultados obtenidos hasta el momento tienen un carácter principalmente exploratorio y han permitido identificar tanto el potencial de los agentes basados en LLMs para apoyar el razonamiento arquitectónico temprano, como limitaciones relevantes, tales como inconsistencias entre propuestas, variabilidad en las respuestas según el modelo utilizado y una fuerte dependencia de la calidad y adecuación del conocimiento incorporado.

Estos avances sientan las bases para las siguientes etapas del trabajo, orientadas a la consolidación del enfoque multi-agente propuesto y a una caracterización más sistemática de su comportamiento en distintos escenarios de requerimientos.

5. Trabajo futuro

El trabajo futuro se organiza en tres horizontes temporales, que reflejan una evolución progresiva del enfoque propuesto desde su estado actual hacia escenarios de mayor complejidad e integración.

En el corto plazo, el foco estará puesto en el diseño y prototipado de una arquitectura multi-agente que permita coordinar de manera explícita las distintas actividades de razonamiento identificadas en este trabajo. La arquitectura será validada mediante casos de estudio con instancias *human-in-the-loop* para evaluar la utilidad, coherencia y trazabilidad de los resultados generados. Asimismo, se trabajará en la extensión de las capacidades de razonamiento de los agentes para soportar estilos arquitectónicos más complejos, y en la mejora de los mecanismos de comunicación entre los mismos, incluyendo estrategias para incorporar retroalimentación humana.

En el mediano plazo, se espera avanzar hacia la integración con entornos de desarrollo ampliamente utilizados y *pipelines CI/CD*. Esto permitirá evaluar cómo los artefactos generados por los agentes pueden incorporarse de manera más directa en flujos de trabajo reales. En este contexto, se explorarán mecanismos para automatizar la transición desde decisiones y tareas de diseño hacia actividades de implementación, sin que esto implique la generación directa de código, sino mas bien la colaboración con desarrolladores humanos o con agentes de generación de código.

En el largo plazo, el objetivo es extender el enfoque hacia un soporte más amplio del ciclo de vida del software, abarcando no solo el *backend* sino también aspectos de *frontend* y la coordinación entre equipos *cross-functional*. En este horizonte, se contempla el desarrollo de entornos de simulación que permitan evaluar el comportamiento de arquitecturas multi-agente bajo escenarios complejos, así como el avance hacia ecosistemas de agentes basados en LLMs con mayor autonomía, capaces de operar de forma robusta en escenarios reales de desarrollo.

Referencias

Ahmad, A., Waseem, M., Liang, P., Fahmideh, M., Aktar, M. S., and Mikkonen, T. (2023). Towards human-bot collaborative software architecting with chatgpt. *arXiv:2302.14600*.

Bachmann, F., Bass, L., Klein, M., and Shelton., C. (2005). Designing software architectures to achieve quality attribute requirements. *IEE Proceedings - Software*.

Bass, L., Clements, P., Kazman, R., and a. O. M. C. Safari (2021). *Software Architecture in Practice, 4th Edition.* Addison-Wesley Professional.

Capilla, R., Jansen, A., Tang, A., Avgeriou, P., and Babar, M. (2016). 10 years of software architecture knowledge management: Practice and future. *Journal of Systems and Software 116*.

Dhar, R., Vaidhyanathan, K., and Varma, V. (2024). Can llms generate architectural design decisions?-an exploratory empirical study. *arXiv:2403.01709*.

Diaz-Pace, J. A., Tommasel, A., Capilla, R., and Ramírez, Y. E. (2025). Architecture exploration and reflection meet llm-based agents. *2025 IEEE 22nd International Conference on Software Architecture Companion (ICSA-C)*.

Díaz-Pace, J. A., Tommasel, A., and Capilla, R. (2024). Helping novice architects to make quality design decisions using an llm-based assistant. *European Conference on Software Architecture*.

Ferrari, R. and Madhavji, N. H. (2008). Software architecting without requirements knowledge and experience: What are the repercussions? *Journal of Systems and Software*.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan., K. R. (2024). Swe-bench: Can language models resolve real-world github issues? *The Twelfth International Conference on Learning Representations*.

Ozkaya, I. (2024). Can architecture knowledge guide software development with generative ai? *IEEE Software, vol. 40, no. 5*.

Sami, M. A., Waseem, M., Zhang, Z., Rasheed, Z., Systä, K., and Abrahamsson., P. (2024). Ai based multiagent approach for requirements elicitation and analysis. *arXiv:2409.00038*.

Wang, L., Ma, C., Feng, X., Zhang, Z., Yang, H., Zhang, J., Chen, Z., Tang, J., Chen, X., Lin, Y., Zhao, W. X., Wei, Z., and Wen., J. (2024). A survey on large language model based autonomous agents. *Fronts. of Comp. Science*.

White, J., Hays, S., Fu, Q., Spencer-Smith, J., and Schmidt., D. C. (2024). *ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design*. Springer Nature Switzerland.

Yao, S., Zhao, J., Yu, D., Du, N., Shafran, I., Narasimhan, K., and Cao, Y. (2022). React: Synergizing reasoning and acting in language models. *arXiv:2210.03629*.

Zhou, A., Yan, K., Shlapentokh-Rothman, M., Wang, H., and Wang, Y. X. (2023). Language agent tree search unifies reasoning acting and planning in language models. *arXiv:2310.04406*.