RAZONAMIENTO EN GRANDES MODELOS DE LENGUAJE

Jorge A. Pichardo Cabrera, Franco Hernández Piloto, Hivan Cañizares Diaz, Alejandro García González, Ernesto Bárcena Trujillo, Carlos Mauricio Reyes Escudero
UH, MATCOM, CC, C411

Resumen: Este trabajo presenta una exploración del estado del arte en razonamiento con Grandes Modelos de Lenguaje (LLMs), destacando resultados clave y las tendencias más relevantes en la materia. Adicionalmente, se llevan a cabo experimentos prácticos utilizando LLMs, con un enfoque en las técnicas de ICL (In-Context Learning) y Prompt Engineering. Finalmente, se propone una estrategia para mejorar la consistencia en el razonamiento de LLMs, basada en el uso de Aprendizaje por Refuerzo.

Introducción

Este trabajo aborda la exploración y mejora del razonamiento en grandes modelos de lenguaje (LLMs) mediante un enfoque estructurado en tres etapas. En primer lugar, se presenta una revisión del estado del arte con un objetivo didáctico, buscando exponer a un alto nivel de abstracción las técnicas más comúnmente utilizadas para optimizar el desempeño de los LLMs en tareas de razonamiento.

Posteriormente, se procede a la experimentación práctica con la API de Gemini de Google. En esta sección, se ponen a prueba las ideas y conceptos extraídos de la revisión del estado del arte, aplicando las técnicas de aprendizaje en contexto y diseño de prompts a la resolución de tareas de razonamiento con Gemini.

Finalmente, se propone una técnica basada en Aprendizaje por Refuerzo (RL) que emplea el seguimiento del código para mejorar la consistencia de las cadenas de razonamiento generadas por los LLMs. Esta estrategia busca aumentar la fiabilidad y la coherencia de los resultados, permitiendo que los LLMs proporcionen soluciones más robustas y confiables.

Revisión del state-of-the-art

A continuación, técnicas relevantes que rescatamos:

scratchpad[1]: Los grandes modelos de lenguaje (LLMs) son buenos en tareas sencillas de 'una sola pasada', pero fallan en tareas que requieren múltiples pasos de cálculo, como sumar números grandes o ejecutar programas complejos. Sin embargo, si se les pide que hagan esas tareas 'paso a paso', mostrando los resultados de cada paso intermedio en una especie de 'hoja de borrador' (scratchpad), los LLMs pueden realizar esos cálculos complejos de manera mucho más efectiva, incluso con pocos ejemplos.

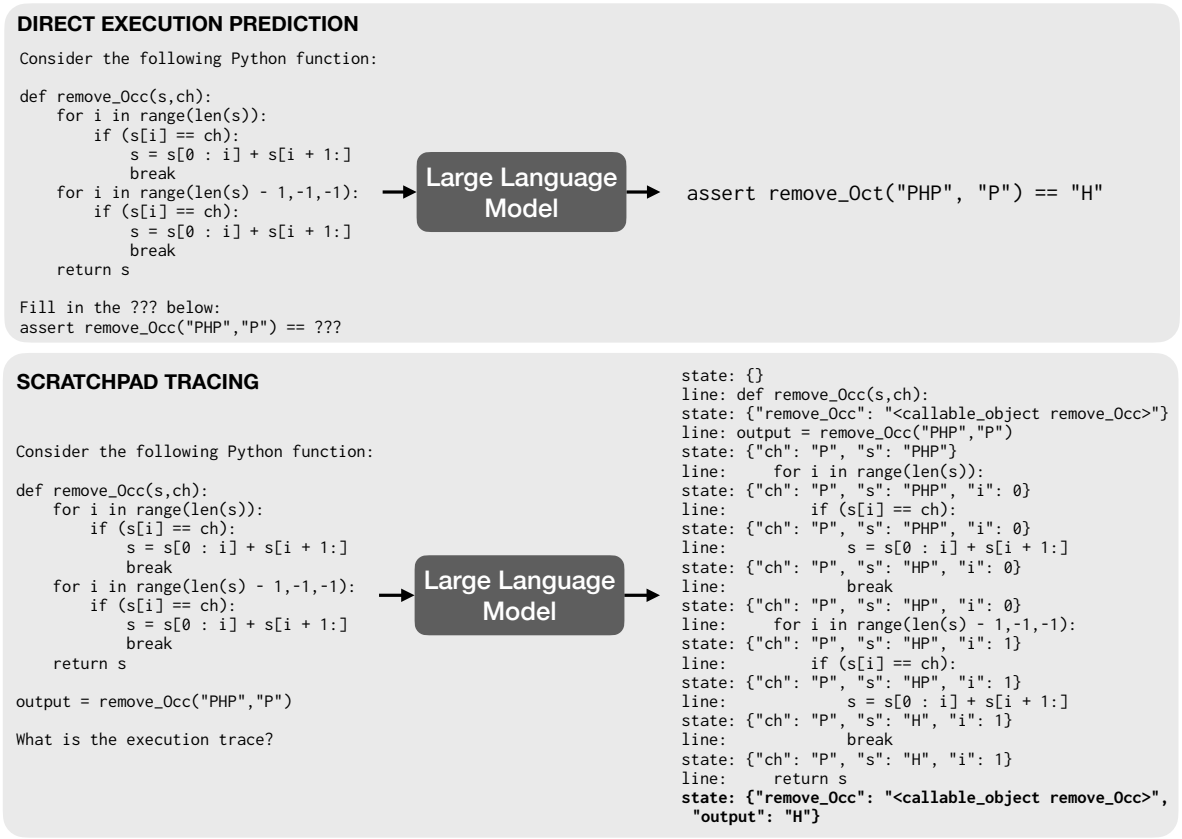

Figura 1: *Ejemplo de aplicación de scratchpad a tarea de predicción de ejecución de código.*

Chain of Thought(CoT)[2]: La clave para mejorar el razonamiento complejo en grandes modelos de lenguaje (LLMs) es guiarlos para que generen una 'cadena de pensamiento' (chain of thought). Esto significa que, en lugar de solo dar la respuesta final, el modelo debe producir una serie de pasos intermedios de razonamiento. Esta capacidad surge de forma natural en LLMs lo suficientemente grandes utilizando una técnica llamada 'chain of thought prompting', donde se les muestran unos pocos ejemplos de cómo generar estas cadenas de pensamiento en los prompts.

Muchos de los consecutivos acercamientos incluyen

mejoras a esta idea inicial.

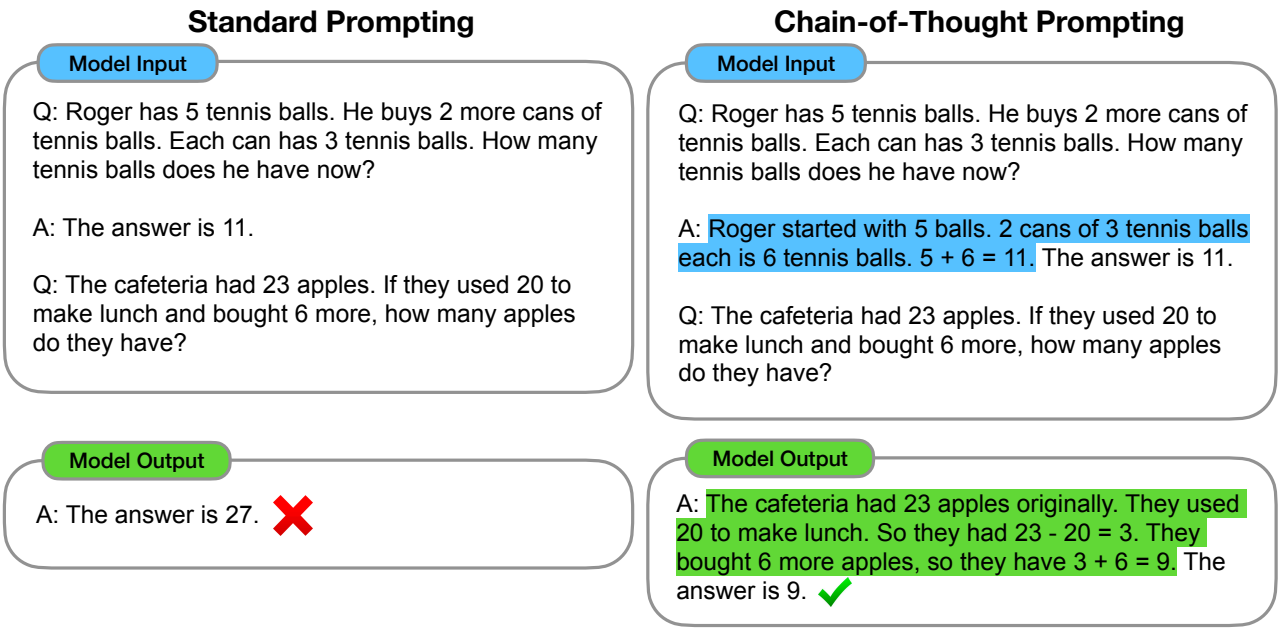

Figura 2: *CoT en tarea de razonamiento matemático.*

Zero-Shot CoT[3]: Los grandes modelos de lenguaje (LLMs) no solo son buenos aprendiendo con pocos ejemplos (few-shot), como se creía, sino que también son capaces de razonar de manera efectiva sin ejemplos, es decir, 'zero-shot', con una sencilla modificación en el prompt. La clave es agregar la frase 'Pensemos paso a paso' antes de que el modelo genere la respuesta. Esta técnica, llamada 'Zero-shot-CoT' (Zero-shot Chain of Thought), hace que el LLM genere una cadena de pensamiento, es decir, que explicite los pasos intermedios del razonamiento, lo cual mejora drásticamente su rendimiento en tareas complejas.

Complexity-Based Prompting[4]: Partiendo de la base de que el 'Chain of Thought' (CoT) mejora el razonamiento en LLMs al generar pasos intermedios, el 'prompting basado en complejidad' propone un refinamiento: seleccionar ejemplos CoT para el prompt y respuestas generadas por el modelo, no solo por su validez, sino por la cantidad de pasos de razonamiento. Al elegir ejemplos más complejos para el prompt (es decir, con más pasos), y al favorecer las respuestas provenientes de cadenas de pensamiento más elaboradas, esta técnica guía al modelo a descomponer los problemas de manera más profunda y a generar soluciones más robustas. Esto se realiza tanto en el proceso de prompting, seleccionando los ejemplos del prompt, como en la decodificación, favoreciendo las respuestas que provienen de razonamientos más complejos, mostrando un aumento de la efectividad del LLM en tareas de razonamiento.

Self-Ask[5]: Propone un método que supera el 'chain of thought' (CoT). En 'self-ask', el modelo se hace preguntas de seguimiento (y las responde) antes de responder a la pregunta inicial. Además, se demuestra que este tipo de prompting estructurado permite integrar un motor de búsqueda para responder a las preguntas de seguimiento, lo que mejora aún más la precisión.

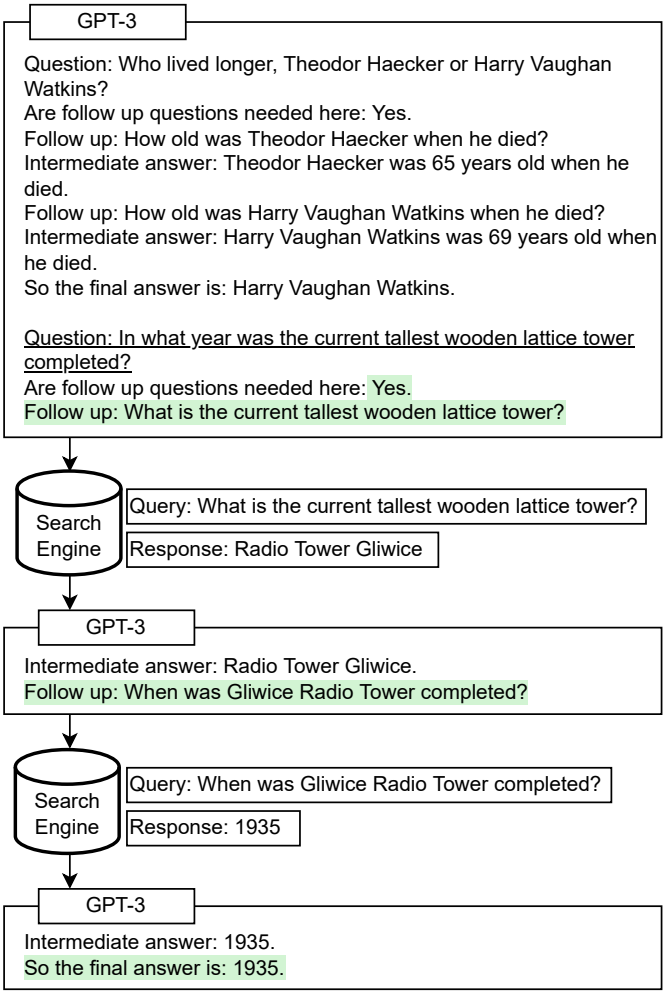

Figura 3: *Flujo de Self-Ask.*

Self-Verification[6]: En un avance reciente sobre las técnicas de razonamiento en grandes modelos de lenguaje (LLMs), se ha explorado la capacidad de los modelos para la 'auto-verificación'. Reconociendo que, aunque el 'chain of thought' (CoT) facilita el razonamiento, sigue

siendo propenso a errores, se ha propuesto un método donde el LLM revisa sus propias respuestas. Este proceso toma la conclusión obtenida mediante CoT y la usa como una condición adicional del problema original. Luego, realizando una verificación 'hacia atrás', el modelo asigna una puntuación a la validez de cada respuesta, seleccionando aquella que obtiene la puntuación más alta, logrando así mejorar la precisión en tareas complejas de razonamiento.

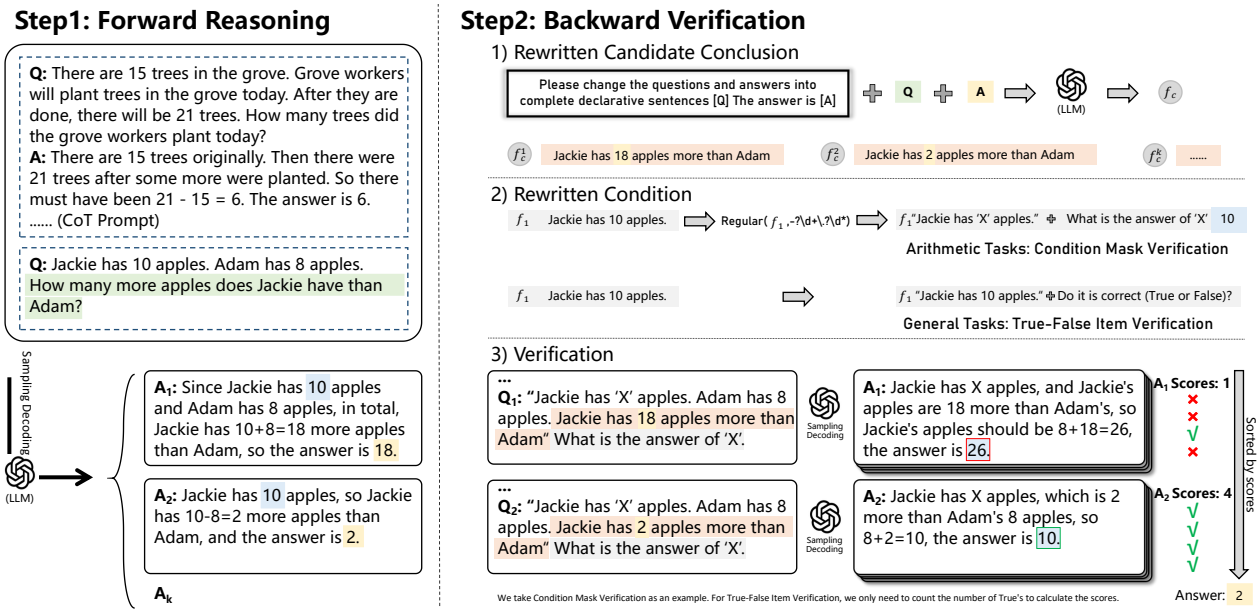

Figura 4: *Self-Verification.*

Self-Consistency[7]: Es una nueva estrategia de decodificación para mejorar el CoT en LLMs. En lugar de utilizar la decodificación 'greedy' tradicional, que solo toma la ruta de razonamiento más probable, la autoconsistencia primero muestrea un conjunto diverso de posibles rutas de razonamiento. Luego, selecciona la respuesta más consistente, evaluando el acuerdo de las respuestas obtenidas a través de las distintas rutas muestreadas. La idea es que un problema complejo de razonamiento suele admitir múltiples maneras válidas de pensarlo que llevan a una única respuesta correcta.

Self-Debugging[8]: El Auto-Depurado es una técnica que capacita a los grandes modelos de lenguaje (LLMs) para depurar autónomamente su código mediante un proceso de auto-análisis 'few-shot'. Al igual que un programador que usa la técnica del 'rubber duck debugging' explicándose a sí mismo el código para encontrar errores, el LLM, mediante el examen de los resultados de ejecución y la auto-explicación del código en lenguaje natural, puede identificar y corregir sus errores sin necesidad de retroalimentación humana o mensajes de error.

Program of Thought(PoT)[9]: Es una técnica que aprovecha modelos de lenguaje, especialmente Codex, para formalizar el proceso de razonamiento como un programa de código ejecutable. A diferencia de enfoques donde el modelo realiza directamente los cálculos y operaciones, PoT delega la computación a un sistema externo, como un intérprete o un entorno de ejecución. El LLM se enfoca en la generación de un programa que describe la lógica de razonamiento y la serie de operaciones necesarias para resolver un problema, pero no ejecuta directamente el programa. En cambio, un ordenador externo es el encargado de procesar y ejecutar este programa generado por el modelo para derivar la respuesta final. En esencia, el LLM se convierte en un diseñador de procesos de computación, formalizando el método de resolución como un programa de código, mientras que la computación en sí misma se externaliza a un entorno más apropiado para su ejecución precisa (ver también Program-aided-language (PAL)[10]).

REFINER[11]: Con el fin de mejorar su razonamiento mediante la generación explícita de inferencias intermedias, como en el 'chain-of-thought' (CoT) y reconociendo que estas inferencias intermedias pueden ser incorrectas, REFINER introduce un 'modelo crítico' que proporciona retroalimentación automatizada sobre el razonamiento. En concreto, el modelo crítico ofrece una retroalimentación estructurada que el modelo de razonamiento utiliza para mejorar iterativamente sus argumentos intermedios.

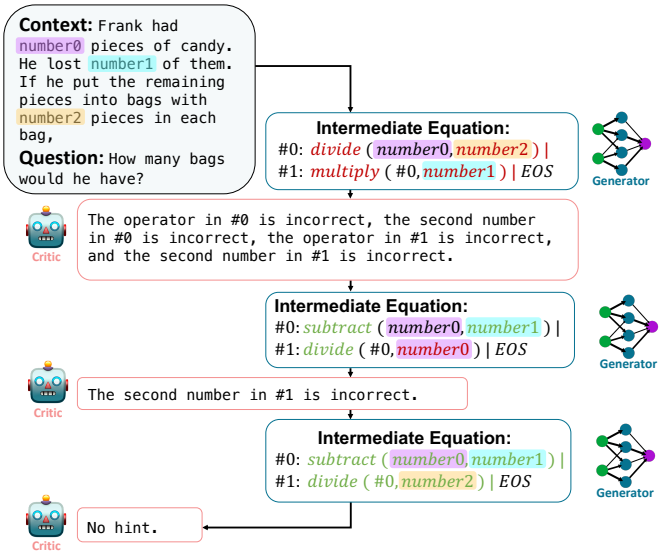

Figura 5: *REFINER*

Ver también 'Self-Correct' en [12] para un acercamiento similar.

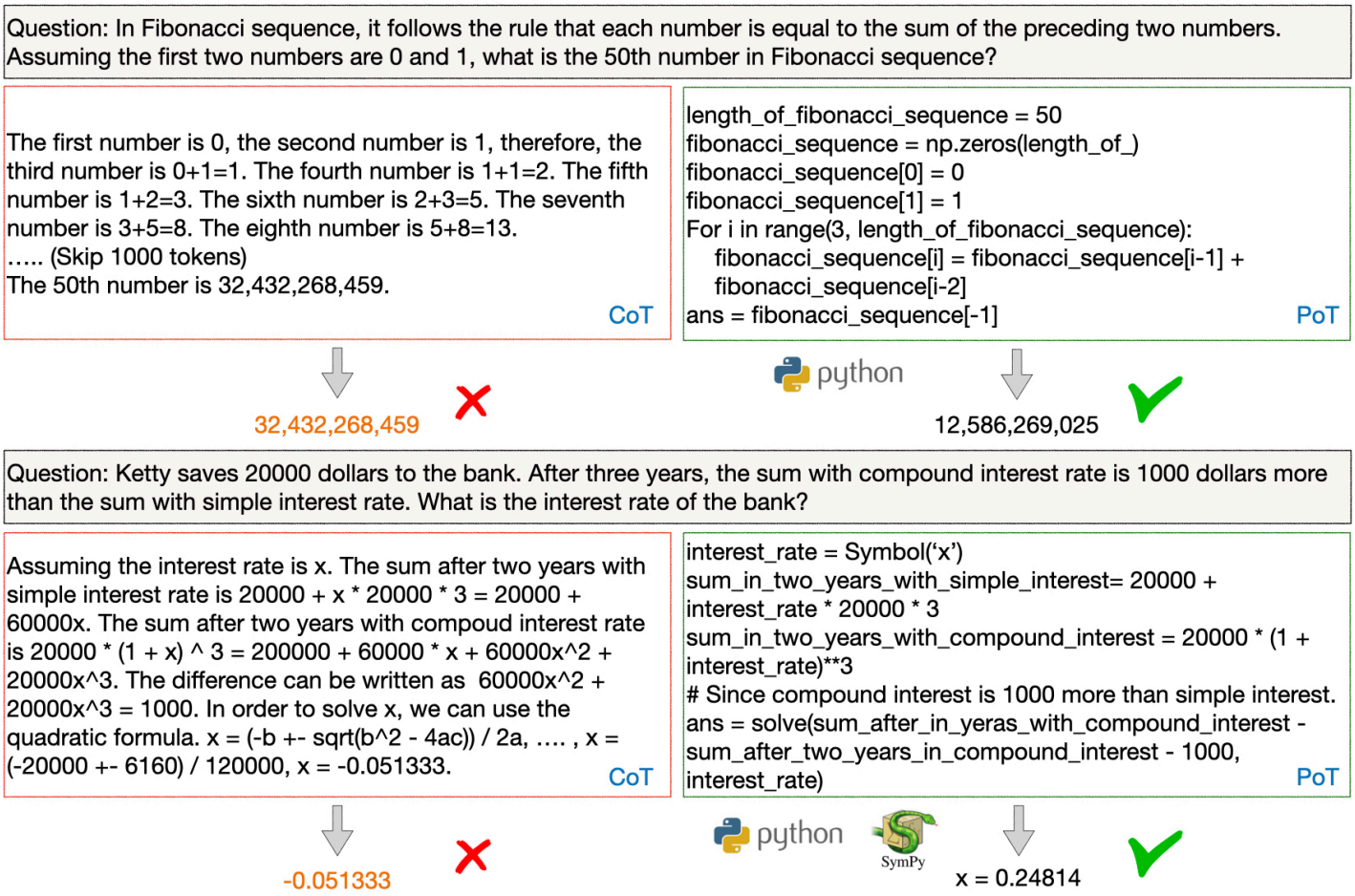

Figura 6: *PoT and PAL.*

Self-Improvement[13]: Este trabajo demuestra que un LLM puede auto-mejorarse utilizando únicamente conjuntos de datos sin etiquetar. Se usa un LLM preentrenado para generar respuestas 'de alta confianza' con razonamiento aumentado para preguntas sin etiquetar, utilizando 'chain-of-thought' (CoT) y autoconsistencia. Luego, se ajusta el LLM utilizando estas soluciones autogeneradas como salidas objetivo. Este enfoque de auto-aprendizaje con datos sin etiquetar mejora las capacidades de razonamiento del LLM y permite alcanzar resultados de última generación en diversos benchmarks. La clave está en que el LLM aprende de sí mismo, guiándose de sus propias respuestas, pero solo de las que genera con una 'alta confianza'.

Self-Refine[14]: Es una técnica que permite a los grandes modelos de lenguaje (LLMs) mejorar sus propias respuestas mediante un proceso iterativo de retroalimentación y refinamiento, inspirado en cómo los humanos revisan y mejoran sus escritos. La idea central es usar un mismo LLM para generar una respuesta inicial, luego proporcionar retroalimentación sobre esa respuesta, y usar esta retroalimentación para refinarla iterativamente. Este proceso no requiere datos de entrenamiento supervisados, entrenamiento adicional o aprendizaje por refuerzo, sino que utiliza el mismo LLM como generador, refinador y proveedor de retroalimentación. De esta manera, el propio modelo se encarga de mejorar su salida usando retroalimentación de su propia salida, en un proceso iterativo de auto-mejora.

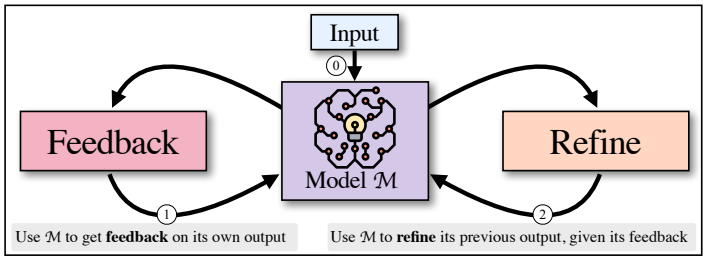

Figura 7: *Self-Refiner*

Progressive Hint[15]: Este trabajo propone simular la capacidad humana de auto-verificación en modelos de lenguaje (LLMs) mediante 'Progressive-Hint Prompting' (PHP). En este método, un LLM genera una respuesta inicial y luego la combina con la pregunta original para una nueva ronda de razonamiento, utilizando su propia respuesta como pista. Si la nueva respuesta coincide con la anterior, se aumenta la confianza en su corrección. El método usa prompts progresivos, que añaden una frase indicando la proximidad de la respuesta a las posibles respuestas anteriores, junto a un recordatorio de dichas respuestas. Este diseño contempla dos situaciones: (1) que las pistas sean iguales a la respuesta correcta, para asegurar que el modelo no se desvíe de la respuesta y (2) que las pistas no sean iguales a la respuesta correcta, para verificar que el modelo puede corregir una respuesta errónea.

Self-Taught Reasoner(STaR)[16] (ojito con este): Es una técnica que permite a los modelos de lenguaje (LMs) mejorar su capacidad de razonamiento mediante un proceso iterativo de auto-aprendizaje. A diferencia de los enfoques que requieren grandes conjuntos de datos con razonamientos explícitos o que sacrifican precisión con pocos ejemplos (few-shot), STaR se basa en un bucle simple. Primero, el modelo genera razonamientos (tipo 'chain-of-thought') para responder preguntas, utilizando unos pocos ejemplos. Si las respuestas son incorrectas, el modelo intenta generar el razonamiento de nuevo, pero esta vez usando la respuesta correcta como guía. Finalmente, se ajusta el modelo con todos los razonamientos que llevaron a la respuesta correcta. Este proceso se repite varias veces, permitiendo que el modelo aprenda a razonar de manera más efectiva a partir de sus propios intentos, usando un mecanismo de auto-aprendizaje para obtener una mejor capacidad de razonamiento (notar que es similar a [13], pero usando 'hints' como en [15]).

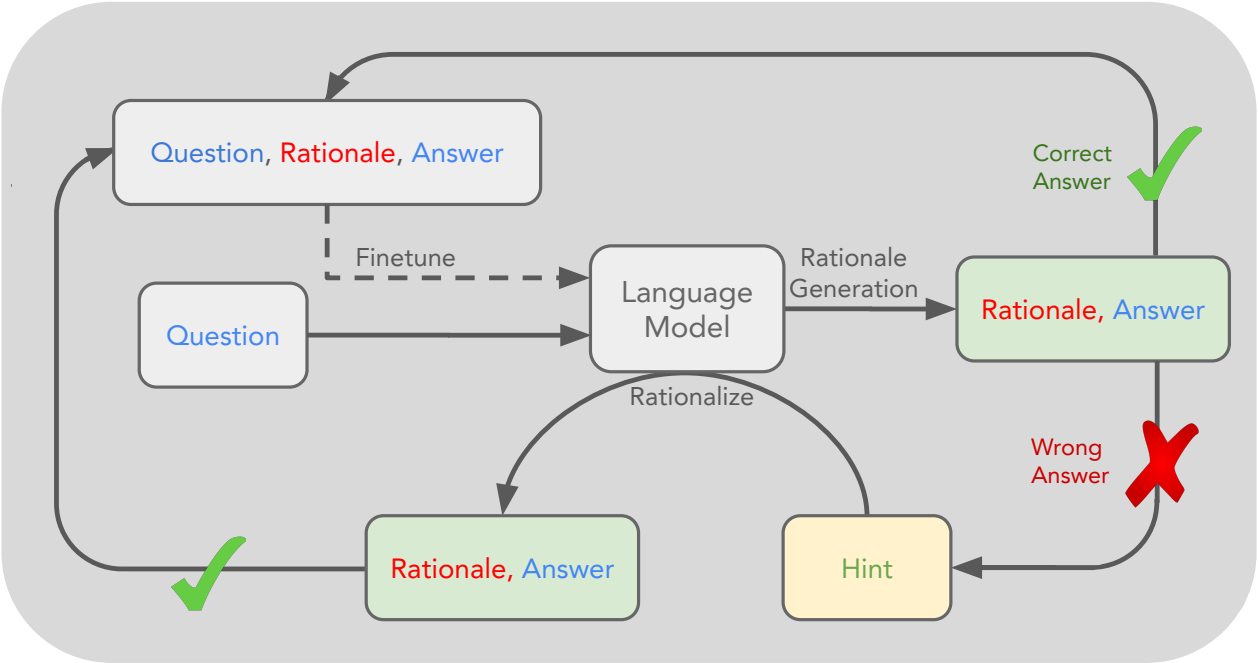

Figura 8: *STaR*

Least-to-most(L2M)[17]: Es una estrategia de prompting que busca solucionar la dificultad del 'chain-of-thought' (CoT) para generalizar a problemas más complejos que los ejemplos usados en el prompt. La idea central es descomponer un problema complejo en una secuencia de subproblemas más simples y resolverlos en orden creciente de dificultad. La solución de cada subproblema se facilita mediante las respuestas obtenidas en los subproblemas previamente resueltos.

Tree-of-Thoughts(ToT)[18]: Es un nuevo marco para la inferencia en modelos de lenguaje (LMs) que busca superar las limitaciones de los procesos de decisión a nivel de token de izquierda a derecha, que dificultan la resolución de problemas que requieren exploración, visión estratégica o decisiones iniciales clave. ToT generaliza el enfoque popular 'Chain of Thought' (CoT) y permite la exploración de múltiples 'pensamientos' (coherent units of text) que actúan como pasos intermedios hacia la solución de un problema. El marco ToT permite a los LMs tomar decisiones deliberadas al considerar diferentes caminos de razonamiento, auto-evaluar sus elecciones y decidir el siguiente paso, permitiendo una búsqueda más estratégica, e incluso retroceder cuando sea necesario para tomar decisiones globales.

Buffer of Thoughts(BoT)[19]: Es un nuevo enfoque de razonamiento aumentado con 'pensamientos' (thought-augmented reasoning) que busca mejorar la precisión, eficiencia y robustez de los grandes modelos de lenguaje (LLMs). BoT utiliza un 'meta-buffer' para almacenar una serie de 'plantillas de pensamiento' (thought-templates) informativas y de alto nivel, que se han extraído de los procesos de resolución de problemas en diversas tareas. Para cada nuevo problema, BoT recupera una plantilla de pensamiento relevante y la adapta con estructuras de razonamiento específicas para llevar a cabo un razonamiento eficiente. Para garantizar la escalabilidad y estabilidad, BoT incorpora un 'buffer-manager' que actualiza dinámicamente el meta-buffer a medida que se resuelven más tareas, aumentando su capacidad.

Consideraciones previas

Aquí incluimos algunos análisis de interés en la construcción de un modelo razonador.

SFT vs RL: Como se analiza en [20], RL, con recompensas basadas en resultados, generaliza mejor que SFT en dominios textuales y visuales, mientras que SFT tiende a memorizar los datos de entrenamiento. Esta diferencia se basa en la intuición de que RL guía al modelo a buscar la forma de plantearse la problemática para alcanzar la solución, mientras que SFT se enfoca en imitar los ejemplos dados. Sin embargo, SFT sigue siendo esencial para estabilizar el entrenamiento de RL y permitir que alcance su potencial de generalización.

Referencias

[1] Maxwell Nye, Anders Johan Andreassen, Guy Gur-Ari, Henryk Michalewski, Jacob Austin, David Bie-

ber, David Dohan, Aitor Lewkowycz, Maarten Bosma, David Luan, Charles Sutton, and Augustus Odena. Show your work: Scratchpads for intermediate computation with language models, 2021.

[2] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. Chain-of-thought prompting elicits reasoning in large language models, 2023.

[3] Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. Large language models are zero-shot reasoners, 2023.

[4] Yao Fu, Hao Peng, Ashish Sabharwal, Peter Clark, and Tushar Khot. Complexity-based prompting for multi-step reasoning, 2023.

[5] Ofir Press, Muru Zhang, Sewon Min, Ludwig Schmidt, Noah A. Smith, and Mike Lewis. Measuring and narrowing the compositionality gap in language models, 2023.

[6] Yixuan Weng, Minjun Zhu, Fei Xia, Bin Li, Shizhu He, Shengping Liu, Bin Sun, Kang Liu, and Jun Zhao. Large language models are better reasoners with self-verification, 2023.

[7] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models, 2023.

[8] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug, 2023.

[9] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks, 2023.

[10] Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models, 2023.

[11] Debjit Paul, Mete Ismayilzada, Maxime Peyrard, Beatriz Borges, Antoine Bosselut, Robert West, and Boi Faltings. Refiner: Reasoning feedback on intermediate representations, 2024.

[12] Sean Welleck, Ximing Lu, Peter West, Faeze Brahman, Tianxiao Shen, Daniel Khashabi, and Yejin Choi. Generating sequences by learning to self-correct, 2022.

[13] Jiaxin Huang, Shixiang Shane Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve, 2022.

[14] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.

[15] Chuanyang Zheng, Zhengying Liu, Enze Xie, Zhenguo Li, and Yu Li. Progressive-hint prompting improves reasoning in large language models, 2024.

[16] Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022.

[17] Denny Zhou, Nathanael Schärli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Claire Cui, Olivier Bousquet, Quoc Le, and Ed Chi. Least-to-most prompting enables complex reasoning in large language models, 2023.

[18] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023.

[19] Ling Yang, Zhaochen Yu, Tianjun Zhang, Shiyi Cao, Minkai Xu, Wentao Zhang, Joseph E. Gonzalez, and Bin Cui. Buffer of thoughts: Thought-augmented reasoning with large language models, 2024.

[20] Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V. Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training, 2025.