Think, Search, Correct: A Self-Reflection Mechanism for Adaptive Tool-Use Agents

Megha Kataki mkataki@ucsd.edu
Arshia Vadhani avadhani@ucsd.edu
Joyce Lu jol072@ucsd.edu
Jiaxin Yang jiy016@ucsd.edu

Kun Zhou kuzhou@ucsd.edu

Abstract

Building upon our Quarter 1 study, we redesign the Search-o1 framework to improve performance, reliability, and factual accuracy in complex multi-hop question answering tasks. Through a prior case study on a subset of the HotpotQA dataset, we identified recurring architectural weaknesses including skipped search calls leading to hallucinated answers, information loss during document extraction, and ineffective multi-hop search reflection.

Unlike traditional Retrieval-Augmented Generation (RAG) systems, Search-o1 iteratively retrieves and analyzes information through a Reason-in-Documents module, dynamically updating reasoning as new evidence is found. However, our earlier evaluations revealed failures in retrieval activation, extraction consistency, and multi-hop planning, resulting in incomplete and unsupported answers.

To address these issues, we introduce a structured self-reflection mechanism consisting of hallucination checks, adaptive query refinement, extraction recovery loops, and constructive content feedback. Our refined architecture enables the agent to iteratively Think, Search, and Correct until sufficient grounded evidence is obtained. We demonstrate how structured reflection and adaptive feedback significantly enhance the reliability of search-augmented language model agents in complex reasoning tasks.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable performance across reasoning and question answering tasks. However, they remain prone to hallucination — the confident generation of incorrect or unsupported information. This issue becomes especially problematic in multi-hop question answering, where answers require synthesizing evidence from multiple sources. In such settings, purely parametric reasoning is insufficient and external retrieval becomes necessary to ensure factual grounding.

Retrieval-Augmented Generation (RAG) systems have emerged as a popular solution by incorporating external documents into the reasoning process. While effective for single-hop retrieval, traditional RAG architectures often struggle with iterative reasoning and dynamic evidence integration.

To address this limitation, our team builds upon Search-o1 Li et al. (2025), a search-enhanced reasoning framework that interleaves chain-of-thought generation with iterative information retrieval. Search-o1 introduces a Reason-in-Documents module that analyzes retrieved content before integrating it into the reasoning stream. Unlike standard RAG pipelines, it dynamically updates its reasoning trajectory as new evidence becomes available.

Despite its architectural strengths, our Quarter 1 case study revealed significant weaknesses. Evaluating Search-o1 on a subset of the HotpotQA dataset using Qwen2.5-3B-Instruct Yang et al. (2024) and the Serper.dev search API, we observed low performance (Accuracy = 0.20, F1 = 0.20). Through detailed log analysis, we identified three recurring failure modes:

- Skipped search calls leading to unsupported final answers.
- Information loss during extraction despite successful retrieval.
- Ineffective multi-hop planning that failed to correct earlier reasoning errors.

These findings suggest that simply interleaving search and reasoning is insufficient. Agents must reflect on retrieved evidence, evaluate gaps, and strategically refine future search actions. Motivated by these observations, we introduce *Think, Search, Correct*, a structured self-reflection framework that incorporates hallucination detection, adaptive query refinement, and constructive feedback mechanisms. Our goal is to design a more self-aware and self-corrective tool-use agent capable of stabilizing multi-hop reasoning and improving factual reliability.

2 Methods

2.1 Adopted Baseline Framework: Search-o1

This study builds upon the Search-o1 architecture Li et al. (2025), a multi-step reasoning framework that interleaves "Chain-of-Thought" generation with iterative information retrieval. The original framework uses a "Reason-in-Documents" module to analyze retrieved information before integrating it into the reasoning stream.

To establish a functional baseline, we modified the backend infrastructure of the original implementation. We retained the Jina API to parse web page content but replaced the deprecated Bing Search API with the Serper.dev (Google) Search API for retrieval tasks. All experiments used Qwen2.5-3B-Instruct Yang et al. (2024) as the backbone Large Reasoning Model.

As illustrated in Figure 1, Visual overview of Search-o1 framework compared to vanilla reasoning pattern and agentic RAG framework, showing the loop between the Large Reasoning

Model, Search Tool, and Reason-in-Documents Module.

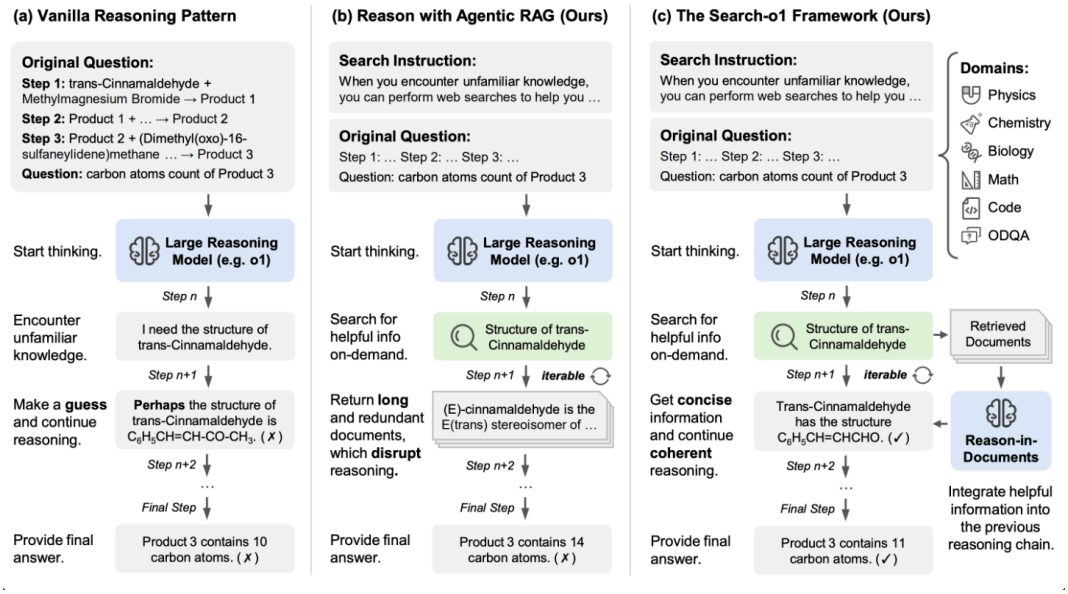

Figure 1: Visual overview of the Search-o1 framework compared to vanilla reasoning and agentic RAG architectures. The system interleaves a Large Reasoning Model, Search Tool, and Reason-in-Documents module in an iterative loop.

2.2 Phase 1: Initial Architecture Design

During our preliminary case study, we identified that the baseline architecture suffered from hallucination, information loss during extraction, and ineffective follow-up queries (poor search planning). To address these limitations, we designed an initial pipeline that incorporates several layers of validation and reflection logic.

2.2.1 Information Extraction Correction

We first addressed a technical bug in the original Search-o1's Reason-in-Documents module, where regex pattern mismatches caused the model to drop information that it had successfully retrieved. We corrected the regex pattern to ensure all generated findings are correctly captured and passed to downstream modules.

2.2.2 Hallucination Detection

To reduce unverified answers, we implemented a verification module. When the model attempts to finalize an answer without prior search, this module scans the generated response for factual claims (e.g., dates, names) that lack citations from the retrieval history. If unverified claims are detected, the system forces the agent to verify the information via

a search cycle. This mechanism prevents the agent from generating confident yet unsupported claims.

2.2.3 Retrieval Judge and Search Query Refinement (Gate 1)

To optimize retrieval relevance, we implemented a "Snippet Judge" to evaluate Serper API results before fetching full documents. This component analyzes retrieved snippets against the current search query to determine if they contain potentially useful leads. If the judge decides the results as irrelevant, it triggers a reflection mechanism prompting the model to refine the search query and search again. This step prevents the agent from wasting inference time and resources on irrelevant documents.

2.2.4 Extraction Refinement Loop

We added an extraction refinement loop to address the issue where the model fails to extract relevant data present in retrieved documents. This loop is triggered when the Reason-in-Documents module returns "No information found." A presence checker scans the raw document text for query-relevant information. If a positive match is found, the system forces the model to re-analyze the documents. This self-correction mechanism recovers lost information and ensures the agent fully utilizes the retrieved information.

2.2.5 Content Judge and Reflection (Gate 2)

Following extraction (in the Reason-in-Documents module), the extracted information passes through a "content judge," which assesses the utility of the extracted findings relative to the immediate search query. The judge evaluates whether the specific information requested by the current search query was successfully found. If the extraction is decided as insufficient, the system appends corrective instructions (e.g., "[System Warning: The information is insufficient. Suggestion: reflection...") to the reasoning stream to guide the next reasoning step.

2.2.6 Judge-Based Routing (Gate 3)

After each search-extraction cycle, we have introduced a high-level routing function aimed at deciding whether the o1-search agent should continue to search or finalize an answer. The judge receives the original question, the agent's current reasoning trace and previous search results, the set of retrieved documents, and the search budget usage. It is prompted to utilize one of three output labels: FINALIZE, SEARCH_MORE, and GIVE_UP following a strict criteria that FINALIZE is only called when the model is certain that the key answer facts is supported by information in the retrieved documents and the reasoning is logical and consistent. If the judge returns SEARCH_MORE, the system appends a structured message (e.g., "The evidence is missing key facts. Generate a NEW and more precise search

query…") that tells the agent to refine the next query and perform another retrieval step in the next turn. If the judge function returns FINALIZE the model will append another system message ("You now have enough information to answer. Do NOT search again. Write the final answer…") pushing the agent to stop searching and create a final answer in the next turn.

2.3 Phase 2: Architectural Refinement and Final Architecture

Our initial diagnostic run and qualitative analysis of the Phase 1 execution logs revealed that some validation gates were often counterproductive. While intended to improve the searching and reasoning process, the judges sometimes introduced inaccurate decisions and conflicts.

2.3.1 Removal of Retrieval and Routing Gates

Retrieval Judge (Gate 1) Removal: Examination of search traces showed that truncated snippets from the search results often lacked enough context for the judge to make an accurate decision. This often led to false negatives, where relevant pages were rejected due to poor snippets/metadata, causing redundant searches. We removed this gate to prioritize recall, allowing the model to fetch the full content of top-ranked search results rather than judging based on incomplete snippets.

Routing Judge (Gate 3) Removal: We observed that the routing classifier frequently made SEARCH_MORE decisions that led to redundant search loops and sometimes conflicted with the model's internal reasoning state. We removed this component to restore model autonomy, allowing the model to determine when it has sufficient information to generate a final answer based on accumulated context. If the model requires further information to reach a final answer, the Gate 2 reflection mechanism provides strategic guidance for the next search step.

2.3.2 Refinement of Content Feedback (Gate 2)

In the phase 1 pipeline, the Content Judge evaluated results strictly, checking whether the extracted information fully satisfied the immediate search query. We observed that this binary approach often caused the model to miss potential candidates or partial clues, providing inaccurate feedback to the agent.

In the final architecture, we refined the judge to evaluate whether the extracted content provides any useful information to the query. This allows the system to classify partial clues (e.g., a new entity to investigate) or useful corrections as successes. If the extracted content is still found to be vague or insufficient, a reflection module analyzes the gap between the user's question and current findings. Unlike the phase 1 version which provided only high-level strategic advice, the refined module formulates a specific strategic follow-up query to guide the agent's next step.

Furthermore, we revised the feedback mechanism to append neutral observations (e.g., "[Observation: The search results were limited. Suggestion: reflection]") rather than the punitive "System Warnings" message. This ensures that the feedback guides the model constructively with a neutral tone without disrupting its reasoning flow.

2.3.3 Final System Architecture

The final optimized pipeline, built upon the Search-o1 baseline, integrates the following components:

- **Hallucination Check:** Hallucination Check: Prevents the generation of unverified factual claims by enforcing validation via a search cycle.
- **Extraction Refinement Loop:** Mitigates information loss by ensuring the model fully utilizes relevant data present in retrieved documents.
- **Content Judge and Reflection (Gate 2):** Analyzes the gap between current search results and user's question, providing constructive feedback to guide subsequent reasoning steps.

3 Results

4 Discussion

5 Conclusion

6 Contributions

Arshia
Attempted to implement the components for the Phase 1 pipeline, including a basic judge and reflection prompting function for content refinement (Gate 3). Worked with Joyce to correct and update the functions in Gate 3 to achieve better overall accuracy and ensure outputted answers met the requirements. Contributed to the report detailing the specifics of Gate 3's judge, reflect, and refine integrations.

Megha
Implemented the initial judge function (Gate 1) and identified the initial placement where the judge function would make a judgment for each document call. Ensured that every document call was properly assessed. Initially created a heuristic approach for evaluation and later transitioned to a Large Reasoning Model (LRM) approach. Contributed to the report's Introduction section.

Joyce

Implemented several components for the Phase 1 pipeline, including the Retrieval Judge and Reflection (Gate 1), the Hallucination Detection module, the Extraction Refinement Loop, and the Content Judge and Reflection (Gate 2). Finalized the architectural refinement for Phase 2, optimizing overall accuracy by identifying and removing bottleneck components (Gate 1 and Gate 3) and refining the Content Judge & Reflection logic (Gate 2). Contributed to the Methods section of the report, documenting the baseline framework and the iterative development process (Phase 1 and Phase 2).

Jiaxin
Implemented the initial reflective functions (Gate 1) and helped refine the prompt organization and reflective reasoning in Phase 1. Merged with Megha's judge function and experimented with a heuristic evaluation approach. Contributed to the report's Abstract section.

References

Li, Xiaoxi, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. "Search-o1: Agentic Search-Enhanced Large Reasoning Models."

Yang, An, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, and ... 2024. "Qwen2 Technical Report."

A Appendix

Below is the original proposal submitted at the beginning of the quarter.

Abstract

We aim to create and develop an adaptive tool-use agent. This agent will strategically reason with itself on when to rely on internal reasoning or to invoke external tools such as a search engine or calculator. LLMs can make strong inferences and exhibit generative abilities, but tend to struggle with tasks that require real-time information retrieval or precise computation, such as a complex calculus problem. Although there are tools that utilize agents to extend LLM capabilities, they often struggle with adaptive decision-making, leading to inefficient tool usage. To address this problem, we want to build open frameworks such as SMART and SMARTAgent, which focus on self-aware and context-sensitive reasoning. We hope to balance internal reasoning with external tool invocation. This approach aims to improve efficiency, accuracy, and flexibility in regards to complex multi-step reasoning situations. This project aims to create a more strategic and self-regulating AI system that is capable of handling complex problems given a certain context effectively.

B Motivation

B.1 Problem Statement

Large Language Models (LLMs) have demonstrated strong reasoning and generative abilities across a wide range of tasks. However, they remain limited when problems require real-time information, factual retrieval, or precise computation. This motivates the integration of external tools, such as search engines or calculators, into LLM-based agents, enabling them to augment internal reasoning with external information retrieval.

Despite progress, existing tool-use agents still lack mechanisms for adaptive decision-making about when and how to invoke tools. In practice, many models often trigger tools based on predefined prompts or implicit patterns learned during training, rather than explicit awareness of their own reasoning state. As a result, agents may overuse tools for simple tasks, underuse them when retrieval is necessary, or fail to revise tool calls when they lead to unhelpful evidence. Therefore, the key challenge lies in enabling LLMs to dynamically balance internal and external reasoning, deciding when to think, when to search, and when to stop.

To build more reliable and efficient tool-use agents, an LLM must be able to determine when internal reasoning is insufficient, decide when to invoke external tools, and evaluate whether the retrieved information supports a correct final answer. Addressing this challenge is essential for building agents capable of reliable long-horizon reasoning and effective tool integration.

B.2 Background, Context, and Limitations

Large Language Models (LLMs) currently face a challenge of over-reliance on external tools, often prioritizing tool use over internal reasoning (Qian et al.). This tendency can lead to inefficiencies, as agents may invoke tools unnecessarily for simple tasks or fail to leverage them appropriately when internal reasoning alone is insufficient. To address this issue, the SMART (Strategic Model-Aware Reasoning with Tools) paradigm was proposed, aiming to optimize task handling by promoting model self-awareness and balanced reasoning strategies. The SMART framework relies on the SMART-ER dataset, which spans three domains and is specifically designed to include reasoning sequences that alternate between internal knowledge and external tool usage. Building on this dataset, SMARTAgent represents a family of models capable of dynamically calibrating their internal and external reasoning approaches depending on task demands. Additional agents and evaluation frameworks, such as SMART-CAL (Shen, Zhu, and Chen) and CuQA (Wang et Al.), have been developed to assess and improve tool-use reliability and adaptive reasoning capabilities. Both tools have seen increases in performance and descreases in error. Recently, Search-o1 (Li et al.)

proposed a multi-step, search-enhanced reasoning pipeline in which an LLM iteratively generates search queries, retrieves documents via search APIs, reasons over documents, and integrates helpful information into its reasoning chain. While Search-o1 achieves strong results on several QA benchmarks, it still lacks explicit mechanisms for self-assessment, query refinement, or determining when tool use is sufficient. Although many advancements have been made, there remains a critical need to optimize both the efficiency and effectiveness of tool usage, particularly for long-horizon tasks that require complex, multi-step reasoning. Current frameworks often lack robust mechanisms for dynamically assessing when tool invocation is necessary or redundant, limiting the practical utility of LLMs in scenarios where strategic reasoning and resource management are essential. They focus primarily on predefined prompts and static answers, disregarding the need for dynamic results especially when necessary for complex, long-horizon, and moving tasks. Addressing these limitations is crucial for developing AI systems that are both generative and strategically capable across a wide range of tasks.

B.3 Case Study Findings

To better understand the limitations of current tool-use agents, we conducted a case study using Search-o1 (Li et al.) on 10 examples from HotpotQA, a multi-hop QA dataset. Our results reveal three systematic error patterns:

- Failure to invoke search when needed: Some queries never triggered a search call, even when external information was essential
- Information loss during extraction: Search results (retrieved documents) contained needed facts, but the model failed to select them as relevant information, causing the reasoning chain to miss correct information.
- Failure to evaluate search quality and ineffective follow-up queries: The model generated unproductive or narrow search queries. When these queries returned irrelevant results, the model failed to revise its previous search query, causing it to repeatedly explore unproductive search directions or return incorrect answers.

These issues expose a gap in Search-o1's ability to reflect on its feedback and adapt its strategy during multi-step tasks, which motivates the need for improved adaptive tool control.

B.4 Objectives

Our project aims to develop mechanisms that improve adaptive tool-use control in LLM agents. The objectives of this proposal are to:

- Develop self-reflection mechanisms that enable the agent to detect unproductive or failed reasoning and search queries, and to refine them into improved versions.

- Introduce a judge mechanism that evaluates whether retrieved evidence is sufficient, decides when additional search is needed, and decides when to finalize an answer.
- Integrate reflection + judging into Search-o1 framework, extending the Search-o1 pipeline so that reflection and judgement occur naturally in the loop, improving reasoning and tool-use stability.
- Create a supplemental "Daily QA" benchmark of practical, real-world questions to evaluate whether the improved agent performs well in realistic scenarios beyond existing QA datasets.
- Evaluate improvements over Search-o1, with metrics including EM/F1 accuracy, number of search calls, and reduction across the three identified failure categories.

C Approach

C.1 Self Reflection Mechanism

C.1.1 Self Reflection Mechanism Steps

1. Recognize that a step was bad: The agent needs to be able to identify that a previous step or search query did not yield relevant or accurate results
2. Analyze the underlying cause: Understand the reason behind the misstep
3. Rewrite a better query: Utilize previous insights from reflection to improve results
4. Repeat until the model succeeds: Continue to iterate until the model results in useful information to provide an accurate answer

C.1.2 Reflection Prompts

General Reflection Prompts: Used to identify any type of failure Ex) Did the previous reasoning step or search query produce useful information? If not, identify why not and suggest an improved alternative.

Error-type Specific Prompts:

- Failure to Search When Needed: Is there a missing piece of information required to answer this question? Suggest a query to identify and answer it.
- Information Loss During Extraction: Did you select all information relevant to the query? If not, identify the missed facts and update your chain of reasoning.
- Poor Follow-up Queries: The previous search query provided irrelevant information. Analyze why and create a revised query better suited to retrieve applicable information.

C.2 Judge Component

C.2.1 System Architecture

- Internal reasoning step(LLM): propose an answer
- Decision module: run reflection prompts to decide if we should finalize the answer, run a search query, or re-reason
- Tool call: if invoked, generates queries and gets the necessary information
- Judge: evaluate whether information is sufficient
- Refinement loop: if evaluation is deemed insufficient, suggest improved queries, repeat until sufficient criteria are met
- Finalize: produce the final answer with cited evidence

C.2.2 Judge Model Design

Input: questions, candidate answer, retrieved evidence, confidence scores, and evidence coverage metric.
Output: Accept, Refine Query, Refine Extraction, Finalize with a summary of rationale
Implementation options:

- Learned Classifier: A Small encoder that is trained on annotated reasoning predicting sufficiency.
- Reranker: Give evidence a score in regards to its relevance to the answer.

C.2.3 Integration with the current Search-o1 pipeline

Extend pipeline by including:

- Reflection after each search and reasoning step
- After extraction, execute the judge function, and again before finalization
- Have a refinement loop that reflects on the output

D Daily Benchmark

D.1 Motivation

Many real world queries contain ambiguous information and needs, incomplete knowledge, and complex contexts.

D.2 Benchmark Design

The Daily QA benchmark will consist of:

- Easy questions collected from every-day questions, mostly could be answered by its pretrained model.
- Tool-necessary queries where the agent must search real-time facts.
- Open-ended decision tasks that require comparing multiple sources of information.
- Fact-check answers with formal citations and references, ensuring we can confirm information with correctness and sufficiency.

Each question will have:

- A set of expected evidence types and keywords
- Criteria for what counts as a "sufficient" answer with self-evaluation of confidence stopping
- A rubric for evaluating whether reasoning and tool-use sequences were appropriate

D.3 Examples

- "What does 'CNN' mean?"
- "Where is Los Angeles located in California?"
- "Which study room is open in Geisel UCSD after 2pm?"
- "What is the current weather in New York City?"
- "Is Data Science a good major or is it bad?"
- "How long is the history of Germany?"
- "What's new in the closest press conference in the White House?"
- "What is the core idea in the transformer paper? I forgot the name"

E Experiments

E.1 Datasets

We will evaluate our approach on open-domain multi-hop QA benchmarks that require models to perform complex reasoning across multiple sources to arrive at an answer. Some benchmarks that we will utilize include:

- HotpotQA: A large scale dataset that requires reasoning across multiple supporting documents to answer questions (Yang et al.)
- 2WikiMultihopQA (2WIKI): A dataset providing explicit reasoning paths for multi-hop questions. (Ho et al.)

- MuSiQue: A dataset containing 2-4 hop questions built from five existing single-hop datasets. (Trivedi et al.)

E.2 Baselines

We will compare our improved agent against the original Search-o1 framework, which serves as the foundation for this project. Search-o1 uses a search-enhanced multi-step reasoning pipeline that iteratively generates search queries, retrieves and reasons over documents, and integrates information into its reasoning chain. By comparing directly against this baseline, we aim to isolate the specific improvements in accuracy and tool-use performance driven by our proposed reflection and judge mechanisms.

E.3 Implementation Details

For the backbone model, we will use the open-sourced Qwen2.5-3B-Instruct. For retrieval, we will use Serper Google Search API, configured with the country set to the United States and the language to English. We will retrieve results from the top two search pages per query. For content extraction, we will use Jina API to fetch the content of web pages from the retrieved URLs. All experiments will be conducted on NVIDIA A30 GPUs.

E.4 Evaluation Metrics

We will evaluate performance based on the following metrics. To assess correctness, we will report Exact Match (EM) and F1 scores to measure the accuracy of the final answers against the ground truth. We will also track tool-use metrics such as the average number of search calls per query, to determine whether our agent utilizes the search tools effectively based on the needs of each task. We will also conduct an error analysis by tracking failure patterns, specifically measuring the reduction across the three identified failure categories in our initial case study.

References

Li, Xiaoxi, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. "Search-o1: Agentic Search-Enhanced Large Reasoning Models."

Yang, An, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, and ... 2024. "Qwen2 Technical Report."