Tool Usage in Large Language Models: Frameworks, Multi-Agent Systems, and Complex Tasks

Andrew Zhu
University of Pennsylvania
andrz@seas.upenn.edu

Abstract

Large Language Models (LLMs) have demonstrated remarkable capabilities in tool usage (also known as function calling), enabling them to interact with external systems and accomplish complex tasks. In this report, I will examine the evolution and current state of tool usage in LLMs through the lens of three of my works at Penn. First, I introduce Kani, a lightweight and model-agnostic framework for LLMs with tool usage that promotes reproducible research. Second, I present ReDel, a toolkit for recursive multi-agent systems that demonstrates how tool usage can enable dynamic task decomposition and delegation between LLM agents. Third, I introduce FanOutQA, a novel benchmark for evaluating complex tool-based information retrieval that requires reasoning across multiple sources. Through extensive experiments, I demonstrate that tool-enabled systems built with these frameworks can outperform traditional baselines across multiple benchmarks while remaining cost-effective. I conclude by discussing future research directions in tool usage, including applications to hierarchical training, shared knowledge representations, and creative domains like storytelling.

1 Introduction

In the modern era of LLMs, LLMs can be trained to request function calls by predicting special tool tokens. The process of a model autonomously deciding to call one or more functions in this manner is called *tool usage* or function calling.

To enable tool usage when a model is trained to do so, the developer must provide the list of tools the model is allowed to use in its prompt. Most tool-enabled models have a specific format they are trained to understand that provides the model with what tools are available, what each does, and the parameters a tool takes as input. When a model predicts that a tool call is appropriate, it can output the name of the tool it wants to use and the inputs to send to that tool, usually expressed in JSON. This is particularly effective as modern LLMs have been pretrained on data containing a large amount of structured data including JSON. Crucially, the model is not generating any code or executing any logic itself – it only outputs a request to run a developer-defined function, and the inputs to pass to that function.

The functions exposed to a tool-enabled LLM are usually human-written, and provide the model additional capabilities that it might otherwise perform poorly on or be incapable of. Some classic examples of tools provided to an LLM are calculators (Andor et al., 2019), document retrieval from a knowledge base (Guu et al., 2020)[1], or internet search (Parisi et al., 2022; Schick et al., 2023). Other tools might enable an LLM to interact with the "outside world" by clicking or typing in a web browser (Yao et al., 2022; Zhou et al., 2024a) or by querying other LLMs (Significant Gravitas, 2023; Shen et al., 2023).

Overall, tool calling not only enables LLMs to perform tasks they would otherwise be incapable of doing, like querying real-time internet data, but also often improves their performance on tasks they could already somewhat do, like complex mathematics.

In this report, we will answer the following questions:

- What properties are desirable in a tool use framework in order to effectively use tools with an LLM in research?
- Can we use LLMs calling other LLMs as tools in order to solve more complex tasks than a single LLM can?
- What kind of complex tasks can we use to evaluate LLMs calling other LLMs as tools?

[1] See Appendix D for further discussion on RAG vs. tool usage.

2 Kani: A Lightweight and Hackable Framework for LLMs with Tool Use

To address the first question, we put forward a non-exhaustive list of desirable properties for a framework for tool-enabled LLMs in research. The framework should 1) allow for reproducible experiments while implementing common boilerplate in an unopinionated way, 2) allow researchers to swap underlying language models with minimal hassle, and 3) be lightweight enough to get started with in less than an hour.

To create a framework that has these properties, we introduce Kani (Zhu et al., 2023b). Kani is a lightweight, flexible, and model-agnostic open-source framework for language models with tool usage. Kani takes care of the basics of chat interaction—querying models, managing chat history, and calling external functions—allowing developers to write robust application code that is interoperable across any underlying language model. From this minimal base, developers can easily override the core features to implement more complex functionality like retrieval, web hosting, dynamic model routing, and tool usage tracking.

Unlike existing frameworks, Kani is lightweight and highly hackable, allowing developers to control their prompts, customize their models, and handle errors with ease. We designed Kani to implement a small number of universally useful core features while providing more complex application-specific examples in extensive documentation. This philosophy allows us to provide a stable foundation that enables researchers to implement complex behavior in their experimental implementation, and rest easy knowing that the underlying software that they use will provide the same experimental setup, every time.

In this section, we show how Kani can be used in conjunction with existing tooling in the LLM space, and contrast Kani to other tool use frameworks. Finally, we will demonstrate how to use Kani with a code sample.

2.1 Kani in the LM Stack

To better contextualize how Kani fits in to the broader ecosystem of software, we lay out our categorization of LM application libraries into a stack four distinct layers: Parameter, Inference, Control, and Application (Figure 1). In this sub-section, we will give a brief overview of what each component of the stack accomplishes, exemplars of software in each stack layer, and how Kani can be used to integrate with each layer.

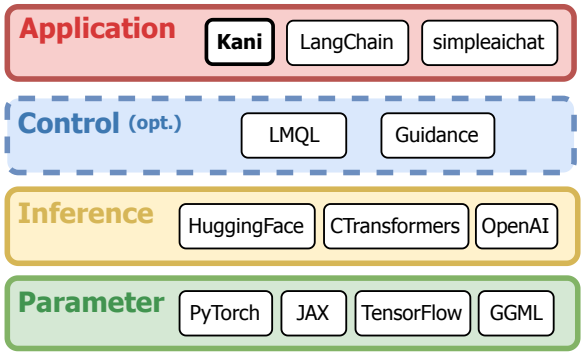


Figure 1: The different layers of the modern LM application stack. Kani sits at the Application layer and is simpler and more flexible than the competing frameworks. Additionally, Kani supports the usage of any lower level control or engine library, allowing developers to use their favorite frameworks alongside Kani.

Parameter Layer. In this layer, LM libraries assist with low-level procedures like matrix operations and hardware acceleration. Examples include PyTorch (Paszke et al., 2019), TensorFlow (Abadi et al., 2016), and JAX (Bradbury et al., 2018). This layer is where the model weights are loaded onto physical devices. Kani is agnostic to the underlying model implementation, so all Model libraries are compatible.

Inference Layer. Libraries like HuggingFace (Wolf et al., 2020) and OpenAI (OpenAI, 2022) in this layer manage elements of model inference such as sampling strategies and tokenization. Kani is interoperable across any Engine library by extending the `BaseEngine` class. In an era characterized by an ever-changing state of the art, the ability to easily swap Engines without changing the application code is invaluable.

Control Layer. Libraries in this optional layer handle complex control logic like dynamic prompt branching and guided decoding. Control libraries include LMQL (Beurer-Kellner et al., 2023) and Guidance (Lundberg et al., 2023). A developer using Kani can use these libraries as middleware between the lower-level LLM and high-level applications, allowing for more robust inference of structured outputs (e.g. JSON).

Application Layer. In the final layer, LM libraries provide the highest level of functionality by managing chat history, compiling prompts, creating

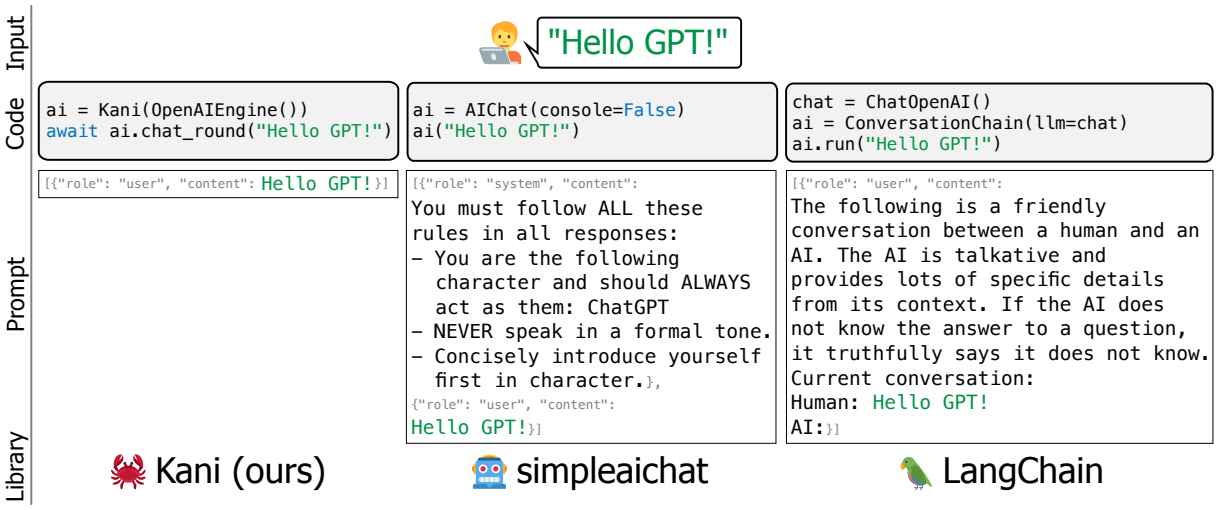


Figure 2: A comparison of prompting behavior between Kani and other competing frameworks. Kani does not edit developers' prompts under the hood in unexpected ways and allows for full control over what is passed to the model.

	Kani	simpleaichat	LangChain
Size (in MB)	13	26	156
Dependencies	2	8	12
Lightweight	✅	✅	❌
Chat Management	✅	❌	❌
Function Retry	✅	❌	❌
Model-Agnostic	✅	❌	✅
Un-opinionated	✅	❌	❌
Extensive Docs	✅	❌	✅

Table 1: A feature comparison between Kani and competing frameworks. Kani is the only package that includes function retrying and chat management while still being lightweight and un-opinionated.

function contexts, and handling errors. Examples of Application libraries include LangChain (Chase et al., 2022), simpleaichat (Woolf and Open Source Contributors, 2023), and, of course, Kani. Kani provides a more flexible, interoperable, and streamlined experience to help any developer build LM applications. In the following section, we compare Kani to these other libraries.

2.2 Features & Framework Comparison

In this section, we compare Kani with simpleaichat (Woolf and Open Source Contributors, 2023) and LangChain (Chase et al., 2022), two popular alternatives that are widely used, to highlight Kani's strengths (see Table 1).

Lightweight. Software like LangChain and simpleaichat, although popular, are built with a "batteries-included" philosophy. This means that it is easy for developers to quickly scaffold and build a functional LLM system using these packages, which include many integrations and default applications. However, this causes subtle issues in their open-source philosophy that can lead to inherent instability when used in downstream research tasks. In the case of LangChain, there are frequent breaking changes driven by the package maintainers' desire to implement cutting-edge integrations with the core package functionality. This means that research code written with a given version of the package might suddenly stop functioning after one of these changes and without warning. This is highly undesirable in scientific research, as this means experiments written using this software may suddenly become unreproducible without notice. Kani is minimalist in both functionality and footprint: we implement essential features with fewer dependencies and less library-specific tooling while accomplishing more (see Table 1). Paired with our detailed documentation, Kani's lean and efficient core of features allows developers to start easily and grow rapidly.

Chat History Management. Unlike our contemporaries, Kani automatically tracks token counts and ensures that the maximum context length is never exceeded—letting developers focus on more exciting parts of their applications. Kani also lets developers easily customize this behavior by overriding default methods in the Kani class.

Robust Function Calling. In contrast to other frameworks, Kani *guarantees* that function calls are valid by the time they reach developers' Python code. If a model calls a function incorrectly, Kani automatically provides feedback to the model and

allows it to try again or follows developers' custom error handling.

Model-Agnostic. Kani provides a straightforward interface to use and interchange *any* model through the `Engine` interface. Developers can easily swap models without altering their source code, simplifying the process of switching models as newer ones are released. Kani will automatically translate abstract tool usage and chat history into the correct underlying prompt for each model.

Un-opinionated Prompting. The "batteries-included" philosophy of LangChain and simpleaichat means that the packages implement hidden prompt hacks in order to produce empirically good outputs (Figure 2). Although this helps developers get off the ground quickly, this kind of behaviour is again undesirable in scientific research. Rigorous science demands that researchers know exactly what inputs are sent to an underlying model, and these packages implement opaque hacks that are hidden from the developer by default. Unlike our contemporaries, Kani does not modify developers' prompts under the hood (see Figure 2). We instead give developers full control to override and construct prompts themselves, leading to more robust, transparent, and reproducible source code.

Extensive Documentation. Kani provides thorough and up-to-date documentation[2] on core library features with a particular focus on customizability. Our docs go beyond basic descriptions of features by including numerous examples of complex applications and guides on how to override and customize default behaviors.

2.3 Tool Usage with Kani

In Kani, tools are written as plain Python methods. There are two ways to provide tools to an LLM with Kani. The primary way is to load them statically by making a subclass of the `Kani` base class and writing class methods with the `@ai_function()` decorator. The other way to incorporate function calling is to load the functions dynamically by passing them in a list to the Kani constructor when instantiating a `Kani` base class or subclass.

In Figure 3, we demonstrate how to create a custom tool and expose it to GPT-4. We define the `WeatherKani` class (L5) and create a `get_weather(loc, unit)` method, which returns the result of some weather API (L6-11). When

[2] https://kani.readthedocs.io/

```
1  class Unit(enum.Enum):
2    FAHRENHEIT = "f"
3    CELSIUS = "c"
4
5  class WeatherKani(Kani):
6    @ai_function()
7    def get_weather(self, loc: Annotated[str,
8    AIParam(desc="The desired city")], unit: Unit):
9      """Get the weather in a given location."""
10     # ... Query some weather API
11     return weather
12
13 engine = OpenAIEngine(api_key, model="gpt-4")
14 chat_in_terminal(WeatherKani(engine))
15 # USER: What's the weather in San Francisco?
16 # AI: [get_weather(
17 #     {"loc": "San Francisco", "unit": "f"})]
18 # TOOL: {"temp": 72, ...}
19 # AI: It's currently 72F in San Francisco.
```

Figure 3: An example showing how to create a subclass of the base Kani and expose a function with `@ai_function`. Functions are given type annotations, triple-quoted docstrings, and `AIParam` descriptions to indicate to the model how they should be used.

the user asks the model for information about the weather (L15), the model generates a request to call this `get_weather` method (L16-17). Kani validates that this call is valid (i.e., it contains all the necessary parameters for the method and they are of the right types) before calling the method. Finally, the method's result is sent back to the model as a continuation of the chat session (L18), and the model generates a grounded completion based on the method's result (L19).

2.4 Tool Prompting

A popular methodology for enabling tool calling in models that aren't explicitly trained for it is through the ReAct prompting framework (Yao et al., 2023). At its core, this framework provides a structured few-shot prompt including a "Think-Act-Observe" loop (Figure 4). In the "Think" step, the model is asked to output a reasoning as to what next step it should take in the pursuit of some goal. In the "Act" step, it outputs a structured string to take that step (usually calling a tool). The tool's result is then returned as an "Observation", and the process repeats until the model decides to terminate with a final result.

Kani fully supports this and other prompting strategies for tool calling. Developers can define

```
USER: What's the weather in Philly?

Thought 1: To find the weather, I should use the
provided get_weather tool.
  Philadelphia is in the United States, which
uses Fahrenheit to measure temperature.
Action 1: get_weather[Philadelphia, Fahrenheit]
Observation 1: cloudy; 75F

Thought 2: I now know the weather in
Philadelphia.
Action 2: Finish[It's currently 75F and cloudy in
Philadelphia, PA.]
```

Figure 4: An example of ReAct prompting. Before calling a tool, the model is prompted to output reasoning ("Thought"). The model requests tool calls with an Action tag, and tools' responses are returned as an Observation.

```python
class KaniWithSummary(Kani):
  @ai_function()
  async def summarize_conversation(self):
    """Get the summary of the conversation."""
    long_context_engine = OpenAIEngine(api_key,
                          model="gpt-4-32k")
    sub_kani = Kani(long_context_engine,
        chat_history=self.chat_history[:-2])
    summary = await sub_kani.chat_round(
    "Please summarize the conversation so far.")
    return summary.content

chat_in_terminal(KaniWithSummary(engine))
# USER: Tell me about trains.
# AI: Trains are modes of long-distance transport
# [Multiple turns of conversation...]
# USER: Summarize the conversation.
# AI: [summarize_conversation()]
# TOOL: [Sub-Kani's LLM response.]
# AI: Our chat began with a general overview
# about trains and how railway systems work...
```

Figure 5: A example showing how to use sub-kani spawning to dynamically delegate summarization to an LLM with a longer context window. Note that the base `"gpt-4"` kani spawns a `"gpt-4-32k"` sub-kani in order to capture the full conversation for summarization.

and supply few-shot prompts and implement structured parsing of model outputs (or use a Control layer library, as seen in Section 2.1). However, it has been shown that prompting-only strategies for tool calling are fragile and susceptible to overreliance on choice of exemplars (Verma et al., 2024). Nevertheless, due to the popularity of these approaches, it is highly likely that future models will include exemplars of such prompts in their training data. Kani allows developers to ensure that these prompts will be formatted correctly now and in the future, while allowing researchers to switch between models pretrained with tool calling capability and prompt-based approaches easily.

2.5 Kani Conclusions

As the field of NLP is advancing rapidly, it is crucial for researchers and developers to ensure that their boilerplate code is robust, reproducible, and does not modify their data in unexpected ways behind the scenes. Although other software packages for interfacing with LLMs exist, they are either locked to a single model, undergo frequent cycles of breaking changes, or modify the developer's input, making them awkward to use in NLP research. Kani is the only library that implements the common boilerplate and offers stability and correctness.

One particularly interesting application of tool calling with Kani is the ability to spawn "sub-Kani" (Figure 5). Sub-Kani are self-contained "agents" capable of performing their own tasks then reporting to their parent with their results. For example, you might have the parent Kani use a cheaper, faster model with a smaller context length. If you need it to perform a task that requires more context, you can spawn a sub-Kani using a more expensive, slower model with a larger context. In the next section, we'll dive deeper into this, and how this enables the creation of a new paradigm of multi-agent systems.

3 Complex Tasks & Agents

With the ability to use tools and interact with the wider world, a popular topic in current work is LLM-powered "agentic" systems. LLM "agents" use tool calling tools across multiple rounds of tool use in pursuit of a goal (usually user-defined) rather than focusing on conversing with a user. In particular, these tools allow the model to perceive (in the form of retrieving information and knowledge) and interact with (e.g., clicking buttons or typing input) with the external world to accomplish the given goal.

For example, a travel planning agent might use various tools to search for flights, restaurants, and attractions in order to help a human user create an itinerary to visit a new city (Xie et al., 2024).

Other agents might interact with a user's computer to fill out a form[3] or shop for products (Yao et al., 2022; Zhou et al., 2024a). Although these tasks are complex as a whole, they are composed of multiple sub-tasks that are accomplishable by a single LLM agent.

Recently, there has been increasing interest in using LLMs to construct complex multi-agent systems to perform these more complex tasks. A multi-agent system uses multiple LLMs together to accomplish complex tasks or answer complex questions beyond the capabilities of a single LLM. In most cases, these systems are defined manually, with a human responsible for defining a static problem-decomposition graph and defining an agent to handle each subproblem in the graph (Hong et al., 2024; Wu et al., 2023; Zhang et al., 2024; Qiao et al., 2024, *inter alia*). Each agent is usually defined with a set persona, "expertise", and tools, in the form of a prompt. This allows each agent to focus on a smaller part of a given complex task which it can complete with a higher success rate (e.g., writing a single routine instead of a whole program, in a code writing task). The human also defines the ways in which each agent communicates with and delegates tasks to other agents.

For example, MetaGPT (Hong et al., 2024) is a system built to write complex Python programs comprised of multiple modules. To accomplish this, the authors define a static system to mimic a human organization. Some example of agents in the system are a product manager agent, responsible for interpreting the human user's request and writing acceptance criterion, a software engineer agent, responsible for generating code, and a quality assurance agent, responsible for determining whether the generated code meets the acceptance criterion. These roles are defined pre-hoc by human experts and do not change at runtime.

An alternative method of using multi-agent systems to complete complex tasks is what we call *recursive* multi-agent systems. A recursive multi-agent system differs from a normal multi-agent system in that the models themselves flexibly decide when to delegate tasks and how to organize their delegation structure by calling tools to spawn new sub-agents (Lee and Kim, 2023; Khot et al., 2023; Prasad et al., 2024), rather than depending on a human-defined set of agents.

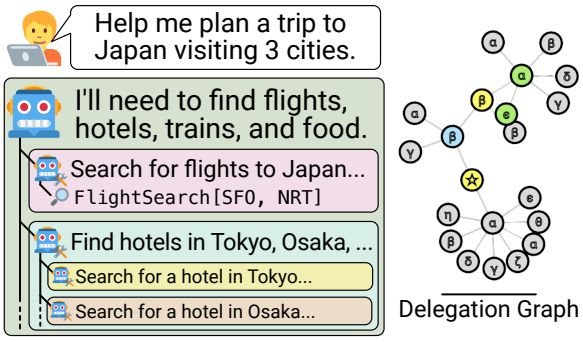


Figure 6: ReDel allows developers to create systems of recursive agents, inspect each agent's state, and visualize a system's delegation graph (right). Recursive agents can be used to solve complex tasks, such as planning a trip to Japan (left).

Many tools and libraries exist for helping create human-defined multi-agent systems, however none support our paradigm of recursive multi-agent systems. In the following sections, I will describe:

- ReDel[4] (Zhu et al., 2024a), a toolkit we built to address this issue and analyze recursive multi-agent systems,
- FanOutQA (Zhu et al., 2024b), a benchmark for evaluating multi-agent systems where the question decomposition is unknown,
- and preliminary results when applying these recursive multi-agent systems to complex tasks.

3.1 Recursive Multi-Agent Systems

In a *recursive* multi-agent system, rather than a human defining the layout of multiple agents, a single root agent is given a tool to spawn additional agents. When faced with a complex task, the root agent can decompose the task into smaller subtasks, then use the provided tool to delegate those tasks to newly-created sub-agents. Each sub-agent can then either complete the task if it is small enough, or recursively decompose and delegate the task further (Figure 6).

In the current landscape of multi-agent systems, the majority of tooling focuses on human-defined static systems, and poorly handles dynamic systems where agents are added to a computation graph at runtime. Furthermore, much of this tooling can be hidden behind paywalls and proprietary licenses. See Table 2 for a detailed comparison between ReDel and similar frameworks.

[3] https://www.anthropic.com/news/developing-computer-use

[4] short for *recursive delegation*

	ReDel	LangGraph	LlamaIndex	MetaGPT	AutoGPT	XAgent
Dynamic Systems	✅	❌	❌	❌	✅	✅
Parallel Agents	✅	✅	✅	❌	❌	❌
Event-Driven	✅	❌	✅	❌	❌	❌
Run Replay	✅	✅	❌	❌	❌	❌
Web Interface	✅	💲	❌	❌	✅	✅
Fully Open Source	✅	❌	❌	✅	✅	✅

Table 2: A feature comparison between ReDel and competing toolkits. ReDel is the only fully open-source toolkit that supports dynamic multi-agent systems with a rich event-driven base and web interface.

```
prompt_toks = Counter()
out_toks = Counter()

for event in read_jsonl("/path/to/events.jsonl"):
  if event["type"] == "tokens_used":
    eid = event["id"]
    prompt_toks[eid] += event["prompt_tokens"]
    out_toks[eid] += event["completion_tokens"]
```

Figure 7: Every event in a ReDel system, builtin or custom, is logged to a JSONL file. Developers can use data analysis tools of their choice to analyze event logs post-hoc. This example demonstrates token counting.

To solve this problem we created ReDel, a fully-featured open-source toolkit for recursive multi-agent systems. ReDel provides the following features in order to help researchers create and analyze these systems:

Modular Interface. ReDel builds on top of Kani (§2) to allow developers to define tools using Python modules. These tools comprise of both functions to expose to LLMs to complete tasks, as well as different methods of delegation to dynamically build a multi-agent system. Each of these modules can be enabled or disabled in a couple lines of code, making it easy to test different system configurations.

Event-Driven System. When systems grow large, it can become difficult to route events from one part to another – for example, how could a developer listen for data from an LLM agent that hasn't even been created yet? ReDel provides a global event-based system that allows developers to listen for LLM events (a list of which is included in Appendix A) as well as define their own custom events. These events are automatically logged to enable post-hoc data analysis with standard data analysis tools. Figure 7 shows how a basic Python script can be used to count a system's token usage post-hoc.

Web Interface. Finally, ReDel features a visual interface (Figure 10) that allows developers to interact with a configured system directly and view replays of saved runs, making it easy to build, iterate on, and analyze the behaviour of recursive multi-agent systems.

We presented ReDel at EMNLP 2024, and are in the process of running additional experiments now. We present preliminary results on three diverse agentic benchmarks below.

3.2 Evaluating Recursive Multi-Agent Systems

With greater capabilities comes the need for more challenging benchmarks. To evaluate ReDel, we began with two existing agentic benchmarks:

1. **TravelPlanner** (Xie et al., 2024): Agents must create travel plans using tools to search flights, restaurant, and attraction databases.
2. **WebArena** (Zhou et al., 2024a): Agents must do complex web tasks such as adding products to a shopping cart or commenting on GitLab.

However, there was one particular kind of question we want these systems to be able to answer that is not tested by these benchmarks. We call these questions "fan-out" questions: natural information-seeking questions that require models to find a list of entities and then consult a large number of documents to aggregate information about those entities to answer a user's question. This pattern of question can be found commonly in day-to-day scenarios, such as performing a literature review (fan-out over research papers), planning a trip (fan-out over attractions), or choosing where to eat.

The fan-out task is particularly challenging for LLMs because it requires multi-hop reasoning across multiple documents, and the combined length of the documents needed to answer the question typically exceeds the length of a model's context window. However, humans can fairly easily reason about the steps needed to solve a fan-out task, but might find it tedious to look at so many sources.

Although there are existing multi-hop question-answering benchmarks (Yang et al., 2018; Talmor

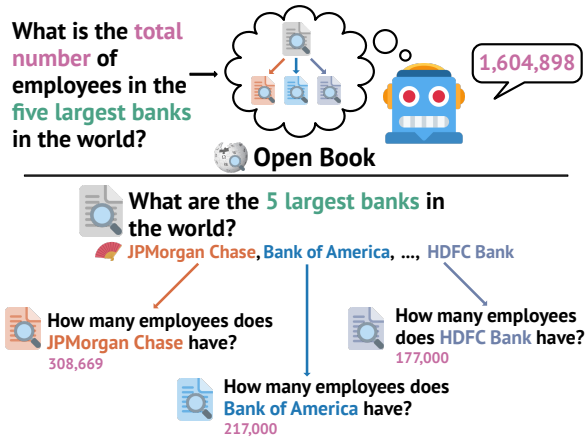


Figure 8: The FanOutQA dataset contains multi-hop, multi-document "fan-out" questions along with human-written decompositions (bottom). LLMs must use tools across multiple rounds of interaction to retrieve multiple Wikipedia articles to answer the question (top).

and Berant, 2018; Ho et al., 2020; Trivedi et al., 2022), we found that these existing benchmarks focus on intra-document dependencies or dependencies between a small number of documents, which does not sufficiently evaluate models' performance on this type of task.

3.2.1 FanOutQA

To cover this gap in question-answering benchmarks, we created FanOutQA (Zhu et al., 2024b).

FanOutQA is a high quality dataset of 1,034 information seeking questions, 7,305 human-written decompositions, and their answers, along with a multi-hop, multi-document benchmark using English Wikipedia as its knowledge base. Compared to other question-answering benchmarks, FanOutQA requires reasoning over a greater number of documents, with its main focus being on the fan-out style of question (Figure 8).

In FanOutQA, we presented three different benchmark settings over the data to evaluate different aspects of LLM systems. In this report, we focus on what we call the *Open Book* setting. The open book setting gives models access to the Wikipedia knowledge base along with the top-level question. The model must use provided retrieval tools to query Wikipedia for relevant articles across multiple rounds of interaction.

For example, consider the question in Figure 8, "What is the total number of employees in the five largest banks in the world?" To answer the question, the model first needs to find a list the five largest banks in the world, which is tabulated on one Wikipedia article. The model would then need to visit each of the five largest banks' articles in order to extract the number of employees at the bank, and finally sum the total to answer the question. We include additional example questions and answers in Appendix C.

Each of these reasoning steps requires using the provided retrieval tool, and might have dependencies on the answer of previous questions. On average, questions in FanOutQA are paired with 172k tokens of evidence spanning 7 documents, making the dataset on par with the other agentic benchmarks mentioned above[5]. We benchmarked multiple single-agent systems on FanOutQA to establish a baseline, the results of which are in Appendix E.

[5]Although RAG does not necessarily imply an agentic task, FanOutQA is formulated as such - see Appendix D for more discussion.

Benchmark	Task	Tools	Example Query
FanOutQA (Zhu et al., 2024b)	Factual question-answering with retrieval tools	Wikipedia Article Search, Retrieve Wikipedia Article	What is the total number of employees in the five largest banks in the world?
TravelPlanner (Xie et al., 2024)	Creation of a structured travel plan that meets hard and commonsense constraints	Search Flights, Calculate Taxi Cost, Search Accommodations, Search Restaurants, Search Attractions, List Cities in State	Please help me plan a trip from St. Petersburg to Rockford spanning 3 days from March 16th to March 18th, 2022. The travel should be planned for a single person with a budget of $1,700.
WebArena (Zhou et al., 2024a)	Interaction with a web browser – mix of information seeking and artifact creation	Click, Type, Hover, Press Key, Scroll, Open New Tab, Change Focused Tab, Close Tab, Go To URL, Go Back, Go Forward	Show me the ergonomic chair with the best rating

Table 3: The benchmarks we used to test ReDel systems, the goal of each benchmark, what tools were provided to the system, and example queries from each benchmark.

System	FanOutQA		TravelPlanner			WebArena		
	Loose	Model Judge	CS-Micro	H-Micro	Final	SR	SR (AC)	SR (UA)
ReDel (GPT-4o)	**0.687**	**0.494**	**67.49**	9.52	**2.78**	0.203	**0.179**	0.643
ReDel (hybrid)	0.551	0.255	65.90	2.14	0.56	0.188	0.171	0.500
ReDel (GPT-3.5-turbo)	0.300	0.087	54.58	0	0	0.092	0.066	0.571
Baseline (GPT-4o)	0.650	0.394	50.83	**18.81**	0	0.162	0.128	**0.786**
Baseline (GPT-3.5-turbo)	0.275	0.077	48.75	0.24	0	0.085	0.058	0.571
Published SotA	0.580	0.365	61.1	15.2	1.11	**0.358**	—	—
Human	0.685	0.452	100	100	100	0.782	0.773	1.000

Table 4: Systems' performance on FanOutQA, TravelPlanner, and WebArena. The SotA models are GPT-4o on FanOutQA, GPT-4-turbo/Gemini Pro on TravelPlanner, and SteP on WebArena. We see that ReDel outperforms the corresponding single-agent baselines across all benchmarks and improves over published SotA in two of three.

3.3 Benchmarking ReDel

Benchmarks. Having created FanOutQA, we thus test ReDel on the three benchmarks mentioned above. In Table 3, we list the benchmarks, the goal of each, and the tools provided to the system.

Due to cost constraints we limited our evaluation to roughly 100–300 examples from each benchmark. We provide the exact query count of each benchmark, example questions, and descriptions of the metrics used in each benchmark in Appendix F.

Models. The ReDel system supports changing the underlying language models used in a plug-and-play manner. For our experiments with ReDel, we used GPT-4o (OpenAI, 2024) and GPT-3.5-turbo (OpenAI, 2022) as the underlying models.

For the two baseline systems, we used the GPT-4o and GPT-3.5-turbo models as-is. All models were given equal access to all tools and no few-shot prompting or fine-tuning was performed.

We also test a hybrid system, where the root agents are powered by GPT-4o and all delegated children are powered by GPT-3.5-turbo.

3.4 Results

Benchmark Performance. In Table 4 we report the results of our evaluation. We see that, across all benchmarks, our recursive delegation system significantly outperforms its corresponding single-agent baseline. We even present an improvement over the previous state of the art systems in both FanOutQA and TravelPlanner.

Furthermore, we see that the gap between ReDel and the baseline system gets larger as the capabilities of the underlying model improves. We believe that this bodes well for the application of such techniques to future, more powerful models.

System	MJ	Δ (↑)	Cost	Δ (↓)
ReDel (4o)	0.494	+25.4%	$222.17	+116%
RD Short (4o)	0.426	+8.2%	$39.84	-61.2%
Base Short (4o)	0.361	-8.2%	$13.71	-86.6%
Baseline (4o)	0.394	—	$102.65	—
ReDel (hybrid)	0.255	+229%	$9.20	+641%
ReDel (3.5-t)	0.087	+12.5%	$2.78	+124%
Baseline (3.5-t)	0.077	—	$1.24	—

Table 5: The cost of running each ReDel system on FanOutQA. Limiting the context of a long-context model to 8192 tokens ("short") can dramatically decrease costs while still showing a performance boost. A hybrid model (i.e., the root agent uses GPT-4o while all delegates use GPT-3.5-turbo) outperforms a small model baseline at the cost of some performance.

Comparison to SotA Systems. In the cases where ReDel fails, namely H-Micro on TravelPlanner and SR on WebArena, these are attributable to metric failures and unequal comparisons. In the TravelPlanner case, on further inspection, we find that recursive systems tend to make more commonsense inputs for meals (e.g. "on the flight" or "packed lunch") – which causes the TravelPlanner evaluation script to give a score of 0 on the Hard Constraint metric. As for the WebArena result, the published SotA SteP model uses few-shot, chain-of-thought prompting, whereas our systems all use zero-shot prompting.

3.5 Cost Analysis

In Table 5, we include the costs of running each system (for API-based models) alongside their performance. The recursive multi-agent systems cost significantly more than a single-agent baseline head-to-head. This is primarily for two reasons. First, multi-agent systems incur communication overhead between agents, although this is a relatively small cost. Second, and primarily, each agent

can retrieve more context to answer a subquestion, fully utilizing its full context window to complete its subtask. Since cost is a function of how many tokens are sent to and generated by a model, multiple models fully utilizing their context windows will naturally exceed the cost of a single model fully utilizing its context window.

Since the primary cost of a recursive multi-agent system is in the utilization of multiple long contexts, we run an additional experiment on the FanOutQA dataset where we limit the context length of each agent to a fraction of its maximum (8192 tokens; 1/16th of max). As seen in Table 5, this can dramatically decrease the cost of the system, while still outperforming the full-length baseline!

We also see that the hybrid model outperforms the small model baselines while still costing far less than a large model baseline. Although the hybrid model loses some performance compared to a large model baseline, it costs far less than using a large model.

3.6 Error Analysis

We also investigated the logs of both the successful runs as well as the failed runs through the replay view of the ReDel web interface (Figure 10d). Through this investigation we observed two common failure cases in recursive multi-agent systems. These cases are as follows:

- **Overcommitment**: The agent attempts to complete an overly-complex task itself.
- **Undercommitment**: The agent performs no work and re-delegates the task it was given.

Overcommitment. We find that overcommitment commonly occurs when an agent performs multiple tool calls and fills its context window with retrieved information. In the ReDel web interface, this manifests as an abnormally small delegation graph, often consisting of only two nodes: the root node, and a single child which the root delegates to and which subsequently overcommits. In practice, this often results in the overcommitting model "forgetting" the task it was meant to accomplish due to the original task being truncated its limited context window. An overcommitting model might fail a task because it outputs a summary of whatever remains in its context window instead of the answer to the original task, whereas a task failure due to causes other than overcommittment might look like a hallucinated result or a simple apology for being unable to complete the task.

Another interesting case of overcommitment is in the TravelPlanner benchmark. This manifested as a single agent using the provided search tools to retrieve flights, attractions, housing, and restaurants. This isn't an issue on its own, as TravelPlanner does not rely on long context reasoning as much as the other two benchmarks. However, if any of the tool calls returned an error (e.g., a specific flight was not found), the model would often abort the plan entirely instead of searching for alternatives for the one part that failed. With a decomposed task, rather than a single agent being responsible for all parts of the plan in addition to flights, there would be a dedicated flight agent. Such an agent would then be able to search for alternative flights if it encountered an error, reducing the amount of context required for the agent and reducing the likelihood of aborting the entire plan.

Undercommitment. In contrast, we find that undercommitment commonly happens when the model incorrectly decides that it does not have the necessary tools to solve the problem and instead assumes that its future child will possess the required tools to solve the problem. In all three benchmarks, this led to failure as agents entered an infinite loop of delegation until they reached a configured depth limit or timed out. In the web interface, this manifests as a line of nodes in the delegation graph (Figure 9).

An illustrative example of undercommitment is in the WebArena benchmark, when an agent delegated a task to click a specific button to a child. The child incorrectly determined that it did not have the capability to click buttons, likely due to pre-training to prevent the model from making false claims about its abilities. As a result, the child delegated the same task to another new child, which delegated to its child, and so on until the system reached the configured maximum depth.

In Table 6 we tabulate the over- and undercommitment rates of ReDel with both GPT-4o and GPT-3.5-turbo for each benchmark. We did this heuristically by counting any delegation graph with two or fewer agents as overcommitted and any delegation graph with a chain of three or more agents with exactly zero or one children as undercommitted. We see that as models get stronger they have a stronger propensity to delegate. However, that propensity to delegate may lead to undercommitment.

System	FOQA		TP		WA	
	OC	UC	OC	UC	OC	UC
RD (4o)	22.7	11.3	41.1	0.5	31.3	44.8
RD (3.5-t)	40.8	1.1	96.7	0	54.6	17.7

Table 6: The overcommitment (OC) and undercommitment (UC) rates, in percent, of the two recursive multi-agent systems we tested, by benchmark.

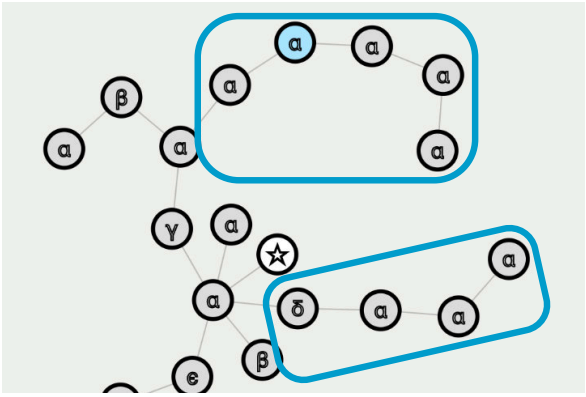


Figure 9: Recursive systems exhibiting undercommitment produce long chains of agents (blue boxes), as seen in the ReDel delegation graph.

Given the prevalence of these two issues, we hypothesize that recursive multi-agent systems may still see further improvements to performance from interventions that target these behaviors. For example, one could fine-tune or prompt agents with domain-specific instructions that detail when the models should delegate and when they should perform tasks on their own.

This case study also helps to demonstrate the strengths of the ReDel system presented above. Using the web interface, it is easy to identify and characterize errors in recursive multi-agent systems. Using ReDel, we will perform more research to further refine such systems.

4 Future Research Directions

Hierarchical Training. In the ReDel work presented above (§3.1), the models we test are simply prompted zero-shot to decompose and delegate subtasks. Although they show impressive ability to do so without any additional training or even few-shot exemplars, unspecialized models still struggle to fully utilize the ability to work together to accomplish a task. I believe a major factor is because models are not developed with a collaborative task-oriented objective in mind during training. ArCHer (Zhou et al., 2024b) shows that a hierarchical RL policy can improve task-oriented performance by incorporating long-horizon task-based rewards into a lower token-level objective. I hypothesize that it is possible to extend this work to incorporate multiple LLM agents. This would allow a global task-level reward to propagate across not just multiple turns of conversation in a single agent, but also to delegated sub-agents. However, this may necessitate additional forms of communication between agents instead of a parent-child relationship, which leads to the next research direction.

Shared Tokens for Recursive Multi-Agent Systems. Currently, each agent in a recursive multi-agent system is self-contained. That is, it is not aware that it is part of a multi-agent system, only receiving tasks from its parent and returning a partial result. This can lead to a lot of duplicated work if two sibling agents have a common dependency - both would have to individually complete the dependency themselves.

To mitigate this, agents could have a shared "blackboard" they could utilize for shared common work (Botti et al., 1995). I hypothesize that rather than having a blackboard at the language level, it is possible to share a vector-level token sink between sibling agents. Given a hierarchical RL policy like the one mentioned above, it might be able to learn abstract task representations in vector space and coordinate between agents.

Better Stories with Tools. One interesting challenge domain for tool usage is tabletop games like Dungeons & Dragons (Martin et al., 2018; Callison-Burch et al., 2022). Games are an important domain for research not just for fun, but because it develops transferrable techniques to real-world problems. However, unlike real-world problems, getting something wrong in a game usually has minimal consequences.

Previous works have shown that structured representations can help story understanding and generation in LLMs (Zhu et al., 2023a). Since then, tool calling has been used to effectively update a dialog state tracker for progression in human-written scenes in tabletop games (Song et al., 2024).

Much of the research on LLM-in-the-loop tabletop games focuses on the LLM acting as a player in the game itself, but I am interested in settings where an LLM only augments a human player's abilities. Particularly in D&D, the Dungeon Master has many different responsibilities during the game that are optional but enhance the experience

for all players. An example of one such responsibility is playing a sound effect to accompany a narrative description of a dragon attack. With a modern full-duplex and tool-calling enabled language model, I believe it is possible to create a system that is capable of listening along to a game and autonomously calling tools to carry out these optional responsibilities.

5 Summary

In this report, we showed the importance of solid software foundations for building modern systems through Kani and ReDel, and how they enable researchers to confidently build and analyze complex LLM systems. We looked at how tool calling enables recursive multi-agent systems as a generalization of human-defined multi-agent systems, and how to evaluate them with realistic benchmarks like FanOutQA. We looked at early results in recursive multi-agent systems, showing that they can be both more performant and cheaper than comparative single-agent systems.

In the future, there are countless directions to expand research on tool usage - and I'm proud that some of the software I've developed has been a pivotal piece of some research projects already (Song et al., 2024; Thudium et al., 2024; Chang et al., 2024). I look forward to continuing my work at Penn and collaborating with talented researchers in NLP, software, and storytelling.

Acknowledgments

The research mentioned in this report is supported in part by the Office of the Director of National Intelligence (ODNI), Intelligence Advanced Research Projects Activity (IARPA), via the HIATUS Program contract #2022-22072200005 and Defense Advanced Research Projects Agency's (DARPA) SciFy program (Agreement No. HR00112520300). This material is based upon work supported by the National Science Foundation Graduate Research Fellowship, under Grant No. DGE-2236662. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or views, either expressed or implied, of ODNI, IARPA, DARPA, the NSF, or the U.S. Government. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation therein.

Thank you to all my co-authors, especially Liam Dugan and Alyssa Hwang. Thanks to Annika Heuser for helping me brainstorm how to take thoughts out of my brain and put them on paper.

Contribution Statement

Many of the works presented in this report are collaborative works, of which I am the first author. In this section, I will list collaborative works and a breakdown of each author's contributions.

- Kani (Zhu et al., 2023b): I wrote the entirety of the Kani library. Liam Dugan, Alyssa Hwang, and I wrote the paper collaboratively. Liam focused on framing and overall structure and Alyssa helped with presentation, figures, and editing.
- ReDel (Zhu et al., 2024a): I wrote the entirety of the ReDel system, ran all experiments, and wrote the paper. Liam Dugan helped with framing, presentation, and editing.
- FanOutQA (Zhu et al., 2024b): I collected the data, and ran all experiments. Alyssa Hwang and I wrote the paper collaboratively.

References

Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, Manjunath Kudlur, Josh Levenberg, Rajat Monga, Sherry Moore, Derek G. Murray, Benoit Steiner, Paul Tucker, Vijay Vasudevan, Pete Warden, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. 2016. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation (OSDI '16)*, pages 265–283, USA. USENIX Association.

Daniel Andor, Luheng He, Kenton Lee, and Emily Pitler. 2019. Giving BERT a calculator: Finding operations and arguments with reading comprehension. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 5947–5952, Hong Kong, China. Association for Computational Linguistics.

Luca Beurer-Kellner, Marc Fischer, and Martin Vechev. 2023. Prompting Is Programming: A Query Language for Large Language Models. *Proceedings of the ACM on Programming Languages*, 7(PLDI):1946–1969.

V. Botti, F. Barber, A. Crespo, E. Onaindia, A. Garcia-Fornes, I. Ripoll, D. Gallardo, and L. Hernández.

1995. A temporal blackboard for a multi-agent environment. *Data & Knowledge Engineering*, 15(3):189–211.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. 2018. JAX: Composable Transformations of Python+NumPy Programs.

Chris Callison-Burch, Gaurav Singh Tomar, Lara J. Martin, Daphne Ippolito, Suma Bailis, and David Reitter. 2022. Dungeons and Dragons as a Dialogue Challenge for Artificial Intelligence. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Allen Chang, Hita Kambhamettu, Neil Sehgal, and Andrew Zhu. 2024. PAL: Palliative assisted learning-bot for clinical interview training. Work in progress.

Harrison Chase, Bagatur, Davis Chase, Zander Chase, Leonid Gandeline, Eugene Yurtsev, and Nuno Campos. 2022. LangChain.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Ming-Wei Chang. 2020. Realm: retrieval-augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org.

Xanh Ho, Anh-Khoa Duong Nguyen, Saku Sugawara, and Akiko Aizawa. 2020. Constructing a multi-hop QA dataset for comprehensive evaluation of reasoning steps. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 6609–6625, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Sirui Hong, Mingchen Zhuge, Jonathan Chen, Xiawu Zheng, Yuheng Cheng, Jinlin Wang, Ceyao Zhang, Zili Wang, Steven Ka Shing Yau, Zijuan Lin, Liyang Zhou, Chenyu Ran, Lingfeng Xiao, Chenglin Wu, and Jürgen Schmidhuber. 2024. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed prompting: A modular approach for solving complex tasks. In *The Eleventh International Conference on Learning Representations*.

Soochan Lee and Gunhee Kim. 2023. Recursion of thought: A divide-and-conquer approach to multi-context reasoning with language models. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 623–658, Toronto, Canada. Association for Computational Linguistics.

Scott Lundberg, Marco Tulio Ribeiro, and Contributors. 2023. Guidance.

Lara J. Martin, Srijan Sood, and Mark O. Riedl. 2018. Dungeons and DQNs: Toward Reinforcement Learning Agents that Play Tabletop Roleplaying Games. In *Joint Workshop on Intelligent Narrative Technologies and Workshop on Intelligent Cinematography and Editing (INT-WICED)*, Edmonton, AB, Canada. http://ceur-ws.org.

OpenAI. 2022. ChatGPT: Optimizing Language Models for Dialogue.

OpenAI. 2024. Hello GPT-4o.

Aaron Parisi, Yao Zhao, and Noah Fiedel. 2022. TALM: tool augmented language models. *Preprint*, arXiv:2205.12255.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An Imperative-Style, High-Performance Deep Learning Library. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems (NeurIPS '19)*, volume 32. Curran Associates, Inc.

Archiki Prasad, Alexander Koller, Mareike Hartmann, Peter Clark, Ashish Sabharwal, Mohit Bansal, and Tushar Khot. 2024. ADaPT: As-needed decomposition and planning with language models. In *Findings of the Association for Computational Linguistics: NAACL 2024*, pages 4226–4252, Mexico City, Mexico. Association for Computational Linguistics.

Shuofei Qiao, Ningyu Zhang, Runnan Fang, Yujie Luo, Wangchunshu Zhou, Yuchen Jiang, Chengfei Lv, and Huajun Chen. 2024. AutoAct: Automatic agent learning from scratch for QA via self-planning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3003–3021, Bangkok, Thailand. Association for Computational Linguistics.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. In *Thirty-seventh Conference on Neural Information Processing Systems*.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. 2023. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. In *Advances in Neural Information Processing Systems*, volume 36, pages 38154–38180. Curran Associates, Inc.

Significant Gravitas. 2023. AutoGPT.

Jaewoo Song, Andrew Zhu, and Chris Callison-Burch. 2024. You have thirteen hours in which to solve the labyrinth: Enhancing AI game masters with function

calling. In *The 4th Wordplay: When Language Meets Games @ ACL 2024*.

Alon Talmor and Jonathan Berant. 2018. The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

Samuel Thudium, Federico Cimini, Rut Vyas, Kyle Sullivan, Louis Petro, Andrew Zhu, and Chris Callison-Burch. 2024. Outwit, outplay, out-generate: A framework for designing strategic generative agents in competitive environments. In *The 4th Wordplay: When Language Meets Games @ ACL 2024*.

Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multi-hop questions via single-hop question composition. *Transactions of the Association for Computational Linguistics*, 10:539–554.

Mudit Verma, Siddhant Bhambri, and Subbarao Kambhampati. 2024. On the brittle foundations of react prompting for agentic large language models. *Preprint*, arXiv:2405.13966.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Max Woolf and Open Source Contributors. 2023. simpleaichat.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W White, Doug Burger, and Chi Wang. 2023. AutoGen: enabling next-gen llm applications via multi-agent conversation. *Preprint*, arXiv:2308.08155.

Jian Xie, Kai Zhang, Jiangjie Chen, Tinghui Zhu, Renze Lou, Yuandong Tian, Yanghua Xiao, and Yu Su. 2024. TravelPlanner: A benchmark for real-world planning with language agents. In *Forty-first International Conference on Machine Learning*.

Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A dataset for diverse, explainable multi-hop question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

Shunyu Yao, Howard Chen, John Yang, and Karthik R Narasimhan. 2022. Webshop: Towards scalable real-world web interaction with grounded language agents. In *Advances in Neural Information Processing Systems*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

Ceyao Zhang, Kaijie Yang, Siyi Hu, Zihao Wang, Guanghe Li, Yihang Sun, Cheng Zhang, Zhaowei Zhang, Anji Liu, Song-Chun Zhu, Xiaojun Chang, Junge Zhang, Feng Yin, Yitao Liang, and Yaodong Yang. 2024. Proagent: Building proactive cooperative agents with large language models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(16):17591–17599.

Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. 2024a. Webarena: A realistic web environment for building autonomous agents. In *The Twelfth International Conference on Learning Representations*.

Yifei Zhou, Andrea Zanette, Jiayi Pan, Sergey Levine, and Aviral Kumar. 2024b. ArCHer: Training language model agents via hierarchical multi-turn RL. In *Forty-first International Conference on Machine Learning*.

Andrew Zhu, Karmanya Aggarwal, Alexander Feng, Lara J. Martin, and Chris Callison-Burch. 2023a. FIREBALL: A dataset of dungeons and dragons actual-play with structured game state information. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 4171–4193, Toronto, Canada. Association for Computational Linguistics.

Andrew Zhu, Liam Dugan, and Chris Callison-Burch. 2024a. ReDel: A toolkit for LLM-powered recursive multi-agent systems. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 162–171, Miami, Florida, USA. Association for Computational Linguistics.

Andrew Zhu, Liam Dugan, Alyssa Hwang, and Chris Callison-Burch. 2023b. Kani: A lightweight and highly hackable framework for building language model applications. In *Proceedings of the 3rd Workshop for Natural Language Processing Open Source Software (NLP-OSS 2023)*, pages 65–77, Singapore. Association for Computational Linguistics.

Andrew Zhu, Alyssa Hwang, Liam Dugan, and Chris Callison-Burch. 2024b. FanOutQA: A multi-hop, multi-document question answering benchmark for

large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 18–37, Bangkok, Thailand. Association for Computational Linguistics.

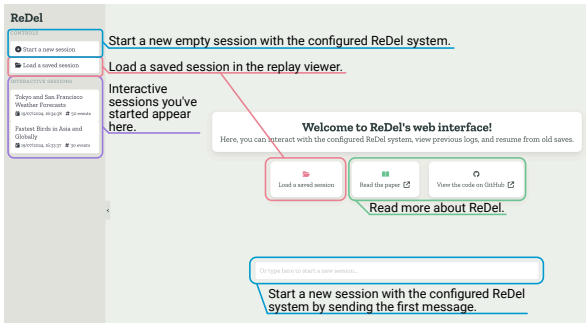
A ReDel Application Events

The following table lists the built-in default events that will be emitted on every run of a ReDel system. Each event has a `type` key which is used to determine what kind of event it is, and a `timestamp` key.

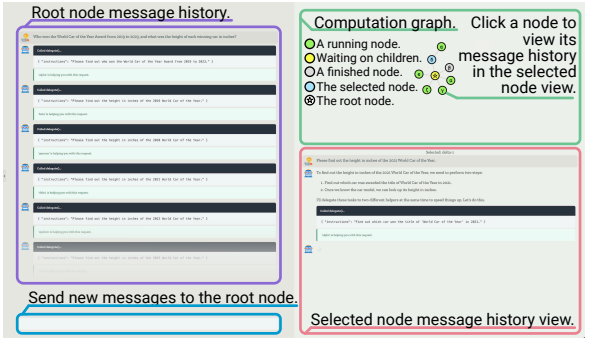
Event Name	Key	Description
Agent Spawned	kani_spawn	A new agent was spawned. The data attached to the event contains the full state of the agent at the time it was spawned, which includes its ID, relations to other agents, a description of the LLM powering it, the tools it has access to, and any system prompts.
Agent State Change	kani_state_change	The running state of an agent changed (e.g. from RUNNING to WAITING). Contains the ID of the agent and its new state.
Tokens Used	tokens_used	An agent made a call to the language model powering it. Contains the ID of the agent, the number of tokens in the prompt it sent, and the number of tokens in the completion the LLM returned.
Agent Message	kani_message	An agent added a new message to its chat history. Contains the ID of the agent and the message's role (e.g. USER or ASSISTANT) and content.
Root Message	root_message	Similar to Agent Message, but only fires for messages in the root node. This is fired in addition to an Agent Message event.
Round Complete	round_complete	Fired when the root node completes a full chat round (i.e. there are no running children and it has generated a response to a user query).

Table 7: A list of events built-in to the ReDel toolkit.

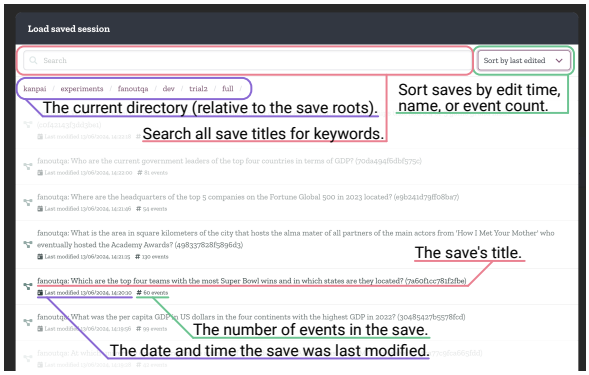
B ReDel Web Interface Screenshots



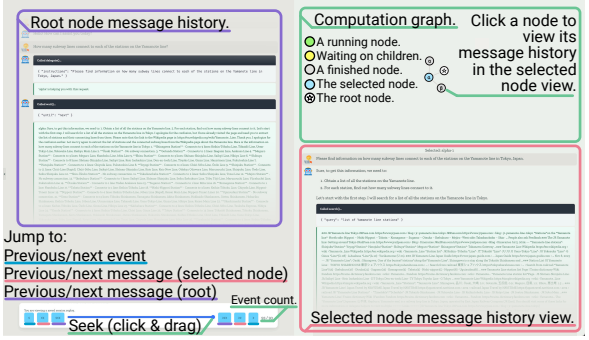
(a) The **home page** of the ReDel web interface.



(b) ReDel's **interactive view** allows users to quickly iterate on prompts and tool design, and test end-to-end performance.



(c) The **save browser** displays logs found in configured directories on the filesystem. It allows developers to search for and review previous runs of ReDel systems.



(d) ReDel's **replay view** allows developers to replay saved runs of ReDel systems, giving events temporal context when analyzing or debugging a system's performance.

Figure 10: The four views of the ReDel web interface: Home (a), Interactive (b), Save Browser (c), and Replay (d).

C FanOutQA Example Questions

In this section, we provide a sample of various questions found in the FanOutQA dataset, along with their human-written decompositions and answers.

1. **Q:** What is the duration in minutes and seconds of the top 5 songs on the Billboard Year-End Hot 100 singles list of 2022?
   Decomposition:
   (a) **Q:** What are the top 5 songs on the list of Billboard Year-End Hot 100 singles of 2022?
   Evidence: https://en.wikipedia.org/wiki/Billboard_Year-End_Hot_100_singles_of_2022
   A: Heat Waves, As It Was, Stay, Easy on Me, Shivers
   (b) **Q:** What is the length of Heat Waves?
   Evidence: https://en.wikipedia.org/wiki/Heat_Waves
   A: 3:58
   (c) **Q:** What is the length of As It Was?
   Evidence: https://en.wikipedia.org/wiki/As_It_Was
   A: 2:43
   (d) **Q:** What is the length of Stay?
   Evidence: https://en.wikipedia.org/wiki/Stay_(The_Kid_Laroi_and_Justin_Bieber_song)
   A: 2:21
   (e) **Q:** What is the length of Easy on Me?
   Evidence: https://en.wikipedia.org/wiki/Easy_on_Me
   A: 3:44
   (f) **Q:** What is the length of Shivers?
   Evidence: https://en.wikipedia.org/wiki/Shivers_(Ed_Sheeran_song)
   A: 3:27

   A: `{"Heat Waves": "3:58", "As It Was": "2:43", "Stay": "2:21", "Easy on Me": "3:44", "Shivers": "3:27"}`

2. **Q:** What are the ages of the top 5 most followed people on Instagram?[6]
   Decomposition:
   (a) **Q:** Who are the top 5 most followed on Instagram?
   Evidence: https://en.wikipedia.org/wiki/List_of_most-followed_Instagram_accounts
   A: Cristiano Ronaldo, Lionel Messi, Selena Gomez, Kylie Jenner, Dwayne Johnson
   (b) **Q:** What is the age of Cristiano Ronaldo?
   Evidence: https://en.wikipedia.org/wiki/Cristiano_Ronaldo
   A: 38
   (c) **Q:** What is the age of Lionel Messi?
   Evidence: https://en.wikipedia.org/wiki/Lionel_Messi
   A: 36
   (d) **Q:** What is the age of Selena Gomez?
   Evidence: https://en.wikipedia.org/wiki/Selena_Gomez
   A: 31
   (e) **Q:** What is the age of Kylie Jenner?
   Evidence: https://en.wikipedia.org/wiki/Kylie_Jenner
   A: 26

[6] As of the dataset epoch of Nov 20, 2023. Retrieved documents return the revision as of this date, so answers are consistent over time.

(f) **Q:** What is the age of Dwayne Johnson?
Evidence: https://en.wikipedia.org/wiki/Dwayne_Johnson
A: 51

A: `{ "Cristiano Ronaldo": 38, "Lionel Messi": 36, "Selena Gomez": 31, "Kylie Jenner": 26, "Dwayne Johnson": 51 }`

3. **Q:** What are the top 4 best-selling mangas of all time and who is the protagonist for each?
Decomposition:

(a) **Q:** What are the top 4 best-selling mangas of all time?
Evidence: https://en.wikipedia.org/wiki/List_of_best-selling_manga
A: One Piece, Golgo 13, Case Closed / Detective Conan, Dragon Ball

(b) **Q:** Who is the protagonist of 'One Piece'?
Evidence: https://en.wikipedia.org/wiki/One_Piece
A: Monkey D. Luffy

(c) **Q:** Who is the protagonist of 'Golgo 13'?
Evidence: https://en.wikipedia.org/wiki/Golgo_13
A: Duke Togo

(d) **Q:** Who is the protagonist of 'Case Closed / Detective Conan'?
Evidence: https://en.wikipedia.org/wiki/Case_Closed
A: Shinichi Kudo

(e) **Q:** Who is the protagonist of 'Dragon Ball'?
Evidence: https://en.wikipedia.org/wiki/Dragon_Ball_(manga)
A: Goku

A: `{ "One Piece": "Monkey D. Luffy", "Golgo 13": "Duke Togo", "Case Closed / Detective Conan": "Shinichi Kudo", "Dragon Ball": "Goku" }`

4. **Q:** Among the Ivy League universities, which four have the lowest endowments and how many Nobel laureates do each of them have?
Decomposition:

(a) **Q:** Which 4 Ivy League universities have the lowest endowment?
Evidence: https://en.wikipedia.org/wiki/Ivy_League
A: Brown University, Dartmouth College, Cornell University, Columbia University

(b) **Q:** How many Nobel laureates does Brown University have?
Evidence: https://en.wikipedia.org/wiki/Brown_University
A: 11

(c) **Q:** How many Nobel laureates does Dartmouth College have?
Evidence: https://en.wikipedia.org/wiki/Dartmouth_College
A: 3

(d) **Q:** How many Nobel laureates does Cornell University have?
Evidence: https://en.wikipedia.org/wiki/Cornell_University
A: 62

(e) **Q:** How many Nobel laureates does Columbia University have?
Evidence: https://en.wikipedia.org/wiki/Columbia_University
A: 103

A: `{ "Brown University": 11, "Dartmouth College": 3, "Cornell University": 62, "Columbia University": 103 }`

5. **Q:** What is the area in square kilometers of the city that hosts the alma mater of all partners of the main actors from 'How I Met Your Mother' who eventually hosted the Academy Awards?
Decomposition:

(a) **Q:** Who are the main actors in 'How I Met Your Mother'?
Evidence: https://en.wikipedia.org/wiki/How_I_Met_Your_Mother
A: Josh Radnor, Jason Segel, Cobie Smulders, Neil Patrick Harris, Alyson Hannigan, Cristin Milioti

(b) **Q:** Which of these actors hosted the Academy Awards?
Evidence: https://en.wikipedia.org/wiki/List_of_Academy_Awards_ceremonies
A: Neil Patrick Harris

(c) **Q:** Who is the partner of Neil Patrick Harris?
Evidence: https://en.wikipedia.org/wiki/Neil_Patrick_Harris
A: David Burtka

(d) **Q:** What is the alma mater of David Burtka?
Evidence: https://en.wikipedia.org/wiki/David_Burtka
A: University of Michigan

(e) **Q:** What city is the University of Michigan in?
Evidence: https://en.wikipedia.org/wiki/University_of_Michigan
A: Ann Arbor, Michigan

(f) **Q:** What is the area of the city of Ann Arbor?
Evidence: https://en.wikipedia.org/wiki/Ann_Arbor,_Michigan
A: 73.35 sq km

A: `73.35 sq km`

6. **Q:** What are the five most popular grape varieties from the Bordeaux appellation, and which area of Bordeaux are they most planted in?
Decomposition:

(a) **Q:** What are the five most popular grape varieties from the Bordeaux appellation?
Evidence: https://en.wikipedia.org/wiki/Bordeaux_wine
A: Cabernet Sauvignon, Cabernet Franc, Merlot, Semillon, Sauvignon Blanc

(b) **Q:** Which area of Bordeaux is Cabernet Sauvignon most planted in?
Evidence: https://en.wikipedia.org/wiki/Cabernet_Sauvignon
A: Haut-Medoc

(c) **Q:** Which area of Bordeaux is Cabernet Franc most planted in?
Evidence: https://en.wikipedia.org/wiki/Cabernet_Franc
A: Saint-Emilion

(d) **Q:** Which area of Bordeaux is Merlot most planted in?
Evidence: https://en.wikipedia.org/wiki/Merlot
A: Saint-Emilion and Pomerol

(e) **Q:** Which area of Bordeaux is Semillon most planted in?
Evidence: https://en.wikipedia.org/wiki/S%C3%A9millon
A: Saint-Emilion

(f) **Q:** Which area of Bordeaux is Sauvignon Blanc most planted in?
Evidence: https://en.wikipedia.org/wiki/Sauvignon_blanc
A: Pessac-Leognan and Graves

A: `{ "Cabernet Sauvignon": "Haut-Medoc", "Cabernet Franc": "Saint-Emilion", "Merlot": "Saint-Emilion and Pomerol", "Semillon": "Saint-Emilion", "Sauvignon Blanc": "Pessac-Leognan and Graves" }`

D Retrieval-Augmented Generation and Tool Use

The relation between retrieval-augmented generation (RAG) and tool use is somewhat subtle; they are often used in conjunction but neither necessarily implies the other. In this report, we focus on RAG as a

use case for tool use. Specifically, this means using tools to search an external knowledge source (e.g., the Internet or Wikipedia) and retrieve text from that source for grounded generation.

In this report, we also use tools to interact with the "outside world" (e.g., clicking a button on a web browser), and use the results of that interaction as grounding for future generations. This type of interaction is not classified as RAG within the scope of this report, as the model is not searching for information – the tool use is intended to select an action in a given action space.

RAG can also be used without tool usage. For example, after a user asks a system some question, the system may retrieve relevant documents from a knowledge base and inject those documents into an LLM's prompt. This would not be considered tool usage, as the model is not requesting the documents itself – the system is designed to always retrieve and inject those documents.

E FanOutQA Single-Agent Performance

In the initial FanOutQA study, we evaluated seven large language models on FanOutQA: GPT-4, GPT-4-turbo, GPT-3.5-turbo, LLaMA 2 70B Chat, Mistral-7B, Mixtral-8x7B, and Claude 2. All models generated text with greedy decoding; all local models were run with FP16 precision. Each model was given tools to search Wikipedia and retrieve articles in order to answer these complex questions.

Most models performed poorly, with the highest-scoring model (GPT-4-turbo) scoring 26.2%. We find this to be because models in this setting "forgot" the original question as their context windows filled with long retrieved passages across multiple retrieval rounds, outputting a summary of the last retrieved passage instead of answering the question. This is supported by a moderate positive correlation between maximum context window sizes and model-judged accuracy ($r^2 = 0.558$). Models with larger context lengths are able to include a greater amount of information in the context and "forget" the original question less often as context windows fill up.

FanOutQA Open Book								
Model	**Ctx Size**	**Loose**	**Strict**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**	**BLEURT**	**GPT Judge**
LLaMA 2 70B	4,096	0.390	0.064	0.157	0.075	0.131	0.443	0.108
GPT-4	8,096	0.315	0.057	0.208	0.106	0.183	0.427	0.164
GPT-3.5-turbo	16,384	0.155	0.032	0.114	0.051	0.099	0.338	0.076
Mistral-7B	32,768	—	—	—	—	—	—	—
Mixtral-8x7B	32,768	0.396	0.055	0.173	0.078	0.147	0.449	0.148
GPT-4-turbo	128,000	0.470	**0.109**	**0.356**	**0.207**	**0.314**	**0.487**	**0.262**
Claude 2.1	200,000	**0.471**	0.086	0.295	0.157	0.253	0.485	0.218
Human	—	0.685	0.289	0.344	0.210	0.307	0.413	0.452

Table 8: Performance of single-agent systems on all metrics in FanOutQA's Open Book setting. We also include human performance in the open-book setting. Mistral-7B's performance is omitted due to catastrophic neural text degeneration.

E.1 Human Performance

We also conducted a human evaluation to create a human baseline and estimate the upper bound of human performance on FanOutQA. We recruited 14 volunteers to each answer 10 FanOutQA questions with access to Wikipedia, similar to the open-book setting. On average, humans took 5–15 minutes to answer each question. In the open-book setting, the humans score significantly higher than our tested models ($p < 0.05$), achieving a loose accuracy of 68.5% and model-judged accuracy of 45.2%. This score may seem low, as the model-judged accuracy does not account for partial credit. As our only automated metric that accounts for partial credit is not robust to typos and equivalent string substitutions, we also manually evaluate the human answers to establish an upper bound of 84.7%.

F ReDel Benchmark Comparison

In Table 9, we tabulate each of the benchmarks tested in our ReDel experiments.

Benchmark	#	Example	Metrics
FanOutQA (Zhu et al., 2024b)	310	What is the total number of employees in the five largest banks in the world?	**Loose**: The average proportion of reference strings found in the generated answer. **Model Judge**: Whether the reference answer and generated answer are equivalent, judged by GPT-4 (`gpt-4-0613`).
TravelPlanner (Xie et al., 2024)	180	Please help me plan a trip from St. Petersburg to Rockford spanning 3 days from March 16th to March 18th, 2022. The travel should be planned for a single person with a budget of $1,700.	**CS-Micro**: The proportion of elements in a generated travel plan that do not demonstrate a commonsense error (e.g. visiting the same attraction twice). **H-Micro**: The proportion of elements in a generated travel plan that do not violate a constraint set by the user or a physical constraint (e.g. budget overruns, non-existent restaurants). **Final**: The proportion of generated travel plans in which there are no exhibited commonsense errors and all constraints are met (i.e., valid travel plans).
WebArena (Zhou et al., 2024a)	271	Show me the ergonomic chair with the best rating	**SR**: Whether the task is successfully completed or correctly marked as unachievable. **SR (AC)**: Whether the task is successfully completed, only among tasks that are achievable. **SR (UA)**: Whether the task is correctly marked as unachievable, only among tasks that are unachievable.

Table 9: The number of queries, example queries, and metrics used from each of the benchmarks we test in the ReDel experiments.