

Intro to Inspect

Open Source Framework for LLM Evals

J.J. Allaire

2024-05-27

Inspect

- A Python package ([inspect_ai](#)) used to create LLM evaluations
- Developed and maintained by the [UK AI Safety Institute](#)
- Similar in function to the eval frameworks embedded in benchmark suites (e.g. Open AI Evals, Eluether LM Eval Harness, etc.) but designed from the ground up for development of more complex evals
- Focus on bridging research and production: provide a great development experience for researchers that results in evals that can be reproducibly run at scale

inspect-llm-workshop

INSPECT ...

CONFIGURATION (.ENV)

- Model Logging
- Model openai (with question mark icon)
- gpt-4-turbo
- Connections default, Retries default, Timeout default

TASK validate ➔ ▶

- Options Task Args
- Limit 100 Epochs default
- Max Tokens default Temperature default
- Top P default Top K default

TASKS

- benchmarks
- arc.py
- arc_challenge
- arc_easy
- gpqa.py
- gsm8k.py
- mathematics.py
- mmlu.py
- honeycomb
- queries.py
- validate
- langchain
- wikipedia.py

queries.py x honeycomb > queries.py > validate

```

26 ➔ Debug Task | ➔ Run Task
27 @task
28 def validate():
29     # read dataset
30     dataset = csv_dataset(
31         csv_file="queries.csv",
32         sample_fields=FieldSpec(
33             input="user_input",
34             metadata=["columns"]
35         ),
36         shuffle=True
37     )
38
39     # create eval task
40     return Task(
41         dataset=dataset,
42         plan=[
43             system_message("Honeycomb AI suggests queries
44             based on user input."),
45             prompt_with_schema(),
46             generate()
47         ],
48         scorer=validate_scorer()
49     )

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- (.venv) jjallaire@JJs-MacBook-Pro inspect-llm-workshop % cd /Users/jjallaire/ukinstitute/inspect-llm-workshop
- (.venv) jjallaire@JJs-MacBook-Pro inspect-llm-workshop % inspect eval honeycomb/queries.py@validate --limit 100 validate (100 samples)

```

total time: 0:00:39
openai/gpt-4-turbo 205,093 tokens [200,216 + 4,877]
accuracy: 0.85
Log: ./logs/2024-05-27T10-13-45_validate_4fTqQtTA7xa6jxZ4UACQW.json

```

Inspect View x

Validate openai/gpt-4-turbo
5/27/2024, 10:13:45 AM – 39 sec

DATASET PLAN SCORER

queries – 100 samples system_message → prompt_with_schema → generate validate_scorer

Samples	Info	Logging	JSON	Scores: All	Sort: sample desc
Open All					
Input	Target	Answer	Score		
100	Show me pods that are crashing	{ "calculatio...ns": ... }	C		
99	recent logs	{"calculati...ons": [{"op": "..."}]}	C		
98	new.correlation	{"breakdow...ns": [...]}	C		
97	calls per second	{"calculati...ons": [{"op": "R..."}]}	I		
96	show me all threads (thread_id) where there was a message with an error and then another...	{"breakdow...ns": ...}	C		

+ ⌂ ⌄ ⌁ ⌃ ⌅ zsh ⌋ Inspect ... ⌋

Ln 38, Col 1 Spaces: 4 UTF-8 LF { Python 3.11.6 ('.venv': venv) ⌋ Prettier ⌋ 3

Core Design

Dataset

List of samples with `input` and `target`

Solvers

Functions that transform dataset inputs, call the model for generation, and act further on model output. Can be composed together as layers, or can be a single layer with higher internal complexity

Scorer

Evaluates final output of solvers. May use text comparisons, model grading, or other custom schemes

Hello, World

```
1 from inspect_ai import Task, eval, task
2 from inspect_ai.dataset import example_dataset
3 from inspect_ai.scorer import model_graded_fact
4 from inspect_ai.solver import (
5     chain_of_thought, generate, self_critique
6 )
7
8 @task
9 def theory_of_mind():
10     return Task(
11         dataset=example_dataset("theory_of_mind"),
12         plan=[
13             chain_of_thought(),
14             generate(),
15             self_critique()
16         ],
17         scorer=model_graded_fact(),
18     )
19
20 eval(theory_of_mind, model="openai/gpt-4")
```

Honeycomb Eval: validate()

```
1 @task
2 def validate():
3     # read dataset
4     dataset = csv_dataset(
5         csv_file="queries.csv",
6         sample_fields=FieldSpec(
7             input="user_input",
8             metadata=[ "columns" ]
9         ),
10        shuffle=True
11    )
12
13    # create eval task
14    return Task(
15        dataset=dataset,
16        plan=[
17            system_message("Honeycomb AI suggests queries based on user input."),
18            prompt_with_schema(),
19            generate()
20        ],
21        scorer=validate_scorer()
22    )
```

Solver: prompt_with_schema()

Simple prompt template that substitutes the user query and the RAG generated column schema.

```
1 @solver
2 def prompt_with_schema():
3
4     prompt_template = resource("prompt.txt")
5
6     async def solve(state, generate):
7         # build the prompt
8         state.user_prompt.text = prompt_template.replace(
9             "{{prompt}}", state.user_prompt.text
10        ).replace(
11            "{{columns}}", state.metadata["columns"]
12        )
13        return state
14
15    return solve
```

Scorer: validate_scorer()

Call the `is_valid()` function w/ the column schema to determine if a valid query was generated.

```
1 @scorer(metrics=[accuracy()])
2 def validate_scorer():
3
4     async def score(state, target):
5
6         # check for valid query
7         query = json_completion(state.output.completion)
8         if is_valid(query, state.metadata["columns"]):
9             value=CORRECT
10        else:
11            value=INCORRECT
12
13        # return score w/ query that was extracted
14        return Score(value=value, answer=query)
15
16    return score
```

validate openai/gpt-4-turbo

5/27/2024, 12:10:53 PM— 1 min 16 sec

accuracy
0.868DATASET
queries — 250 samplesPLAN
system_message → prompt_with_schema → generateSCORER
validate_scorer

Samples	Info	Logging	JSON	Scores: All	Sort: sample asc	Open All
	Input	Target	Answer	Score		
1	slow requests		{"breakdowns": ["http.route"], "calculations": [{"column": "duration_ms", "op": "H..."}]}	C	▼	
2	slow requests		{"breakdowns": ["http.route"], "calculations": [{"column": "duration_ms", "op": "HEATMAP"}]}	C	▼	
3	http status > 500		{"breakdowns": ["http.status_code"], "calculations": [{"op": "COUNT"}], "filters": ...}	I	▼	
4	show me slow requests by endpoint for the last 5 hours		{"breakdowns": ["http.route"], "calculations": [{"column": "duration_ms", "op": "H..."}]}	C	▼	
5	the total length of a trace for each runner-name		{"breakdowns": ["runner-name"], "calculations": [{"column": "duration_ms", "op": "S..."}]}	C	▼	
6	IF(AND(AND(EQUALS(\$service.appprefix, "sfx"), EQUALS(\$service.name, "sfx-polaris-sfc-webservices"), STARTS_WITH(\$http.url, "http://super-performance-returns"))))		{"calculations": [{"op": "COUNT"}], "filters": [{"column": "service.appprefix", "op": "=",...}]} I	I	▼	
7	view each runner-name and the trace that is currently using it		{"breakdowns": ["runner-name", "trace.trace_id"], "calculations": [{"op": "COUNT"}], "filters": ...}	C	▼	
8	Show me errors in the last day, related to SQL		{"calculations": [{"op": "COUNT"}], "filters": [{"column": "db.statement", "op": "exists"}, {"..."}]}	C	▼	
9	at what hour is more used the route /auth/v0/login		{"breakdowns": ["http.target"], "calculations": [{"op": "HEATMAP"}], "filters": ...}	I	▼	
10	slow requests		{"breakdowns": ["http.route"], "calculations": [{"column": "duration_ms", "op": "H..."}]}	I	▼	

validate openai/gpt-4-turbo

5/27/2024, 12:10:53 PM— 1 min 16 sec

accuracy

0.868DATASET
queries — 250 samplesPLAN
system_message → prompt_with_schema → generateSCORER
validate_scorer

Samples	Info	Logging	JSON	Scores:	All	Sort:	sample asc	Close All	Open All
				Input	Target	Answer	Score		
1 slow requests								{"breakdowns": ["http.route"], "calculations": [{"column": "duration_ms", "op": "H..."}]}	
2 slow requests								{ "breakdowns": ["http.route"], "calculations": [{ "column": "duration_ms", "op": "HEATMAP" }...]}	
3 http status > 500								{"breakdowns": ["http.status_code"], "calculations": [{"op": "COUNT"}], "filters": ...}	

Messages Scoring Metadata

system

Honeycomb AI suggests queries based on user input.

user

COLUMNS:status_code,http.response.status_code,status_message,error,http.response.body,server.address,http.route,server.port,exception.type,exception.message,http.response.body.size,client.port,http.response.headers,service.name,http.request.body.size,http.request.method,duration_ms,http.request.body,http.request.headers,client.address,url.query,system.memory.usage_mb,http.forwarded_to,type,name,user_agent.original,url.scheme,span.kind,url.path,enduser.project.name,invocations.count,parent_name,enduser.user.email,subscription.id,enduser.project.id,library.name,enduser.user.id,trace.trace_id,trace.span_id,telemetry.sdk.language,trace.parent_id,enduser.organization.name,span.num_events,span.num_links,meta.signal_type,meta.annotation_type,telemetry.sdk.version,enduser.organization.id,telemetry.sdk.name,db.statement

QUERY SPEC:

All top-level keys are optional.

```
"calculations": [
    // ops: COUNT, CONCURRENCY, COUNT_DISTINCT, HEATMAP, SUM, AVG, MAX, MIN, P001, P01, P05, P10, P25, P50, P75, P90, P95, P99, RATE_AVG, RATE_SUM, RATE_M
    {"op": "COUNT"}, // COUNT and CONCURRENCY are just op
    {"op": "HEATMAP", "column": "name"}
],
"filters": [
    // ops: =, !=, >, <, >=, <=, starts-with, does-not-start-with, exists, does-not-exist, contains, does-not-contain, in, not-in
    {"column": "name", "op": "exists"}, // exists and does-not-exist ops only have column
    {"column": "name", "op": "=", "value": "something"}
]
```

NLQ: by service

EXISTING QUERY:

```
{"calculations": [{"column": "duration_ms", "op": "HEATMAP"}]}, {"breakdowns": ["service.name"], "calculations": [{"column": "duration_ms", "op": "HEATMAP"}]}
```

NLQ: where span is client_request

EXISTING QUERY:

```
{"calculations": [{"column": "duration_ms", "op": "HEATMAP"}], "filters": [{"column": "trace.parent_id", "op": "does-not-exist", "join_column": ""}], "time_range": 604800}, {"calculations": [{"column": "duration_ms", "op": "HEATMAP"}], "filters": [{"column": "trace.parent_id", "op": "does-not-exist", "join_column": ""}, {"column": "name", "op": "=", "value": "client_request", "join_column": ""}], "time_range": 604800}
```

NLQ: add count

EXISTING QUERY:

```
{"breakdowns": ["name"], "calculations": [{"column": "duration_ms", "op": "HEATMAP"}], "filters": [{"column": "http.route", "op": "exists", "join_column": ""}]}, {"breakdowns": ["name"], "calculations": [{"column": "duration_ms", "op": "HEATMAP"}, {"op": "COUNT"}], "filters": [{"column": "http.route", "op": "exists", "join_column": ""}]}
```

NLQ: remove count

EXISTING QUERY:

```
{"breakdowns": ["name"], "calculations": [{"column": "duration_ms", "op": "HEATMAP"}, {"op": "COUNT"}], "filters": [{"column": "service.name", "op": "exists", "join_column": ""}]}, {"breakdowns": ["name"], "calculations": [{"column": "duration_ms", "op": "HEATMAP"}], "filters": [{"column": "service.name", "op": "exists", "join_column": ""}]}
```

NLQ: add count

EXISTING QUERY:

```
{"calculations": [{"op": "COUNT"}]}
```

NLQ: slow requests

EXISTING QUERY:

```
{"breakdowns": ["name"], "filters": [{"column": "error", "op": "exists", "join_column": ""}]}, {"breakdowns": ["http.route"], "calculations": [{"column": "duration_ms", "op": "HEATMAP"}]}
```

NLQ: all pods with name cron-no-medication

EXISTING QUERY:

```
{"calculations": [{"op": "COUNT"}], "filters": [{"column": "k8s.pod.name", "op": "starts-with", "value": "cron-no-medication-observation-received-cron"}, {"join_column": ""}], "filter_combination": "AND", "limit": 100, "time_range": 604800, "granularity": 0}
```

NLQ: http status > 500

EXISTING QUERY:

🤖 assistant

```
{"breakdowns": ["http.status_code"], "calculations": [{"op": "COUNT"}], "filters": [{"column": "http.status_code", "op": ">", "value": "500"}], "orders": [{"op": "COUNT", "order": "descending"}], "time_range": 7200}
```

Honeycomb Eval: critique()

```
1 @task
2 def critique():
3     # read dataset
4     dataset = csv_dataset(
5         csv_file="queries.csv",
6         sample_fields=FieldSpec(
7             input="user_input",
8             metadata=[ "columns" ]
9         ),
10        shuffle=True
11    )
12
13    # create eval task
14    return Task(
15        dataset=dataset,
16        plan=[
17            system_message("Honeycomb AI suggests queries based on user input."),
18            prompt_with_schema(),
19            generate()
20        ],
21        scorer=critique_scorer()
22    )
```

Scorer: critique_scorer()

```
1 @scorer(metrics=[accuracy()])
2 def critique_scorer(model = "openai/gpt-4-turbo"):
3
4     async def score(state, target):
5         # build the critic prompt
6         query = state.output.completion.strip()
7         critic_prompt = resource("critique.txt").replace(
8             "{{prompt}}", state.user_prompt.text
9         ).replace(
10            "{{columns}}", state.metadata["columns"]
11        ).replace(
12            "{{query}}", query
13        )
14
15         # run the critique
16         result = await get_model(model).generate(critic_prompt)
17         parsed = json.loads(json_completion(result.completion))
18         value = CORRECT if parsed["outcome"] == "good" else INCORRECT
19         explanation = parsed["critique"]
20
21         return Score(value=value, explanation=explanation)
22
23     return score
```

critique openai/gpt-4-turbo

5/27/2024, 12:12:32 PM— 54 sec

accuracy
0.58DATASET
queries — 50 samplesPLAN
system_message → prompt_with_schema → generateSCORER
critique_scorer

Samples Info Logging JSON

Scores: All Sort: sample asc Open All

	Input	Target	Answer	Score
1	show respond time heatmap		{"calculations": [{"column": "duration_ms", "op": "HEATMAP"}]}	C ✓
2	i want to search for traces which have graphql.operation.name = GenerateBartFromFileUpload and don't have a span where workflowType=processFileUploadWorkflow		{"calculations": [{"op": "COUNT"}, {"filters": [{"column": "graphql.operation.name": "GenerateBartFromFileUpload", "op": "EQUAL", "value": true}, {"column": "workflowType", "op": "NOT_EQUAL", "value": "processFileUploadWorkflow"}]}]}	I ✓
3	latest traces		{"calculations": [{"op": "COUNT"}, {"orders": [{"op": "COUNT", "order": "descending"}]}]}	I ✓
4	slow requests		{"breakdowns": [{"http.route": true}, {"calculations": [{"column": "duration_ms", "op": "HIGHEST"}]}]}	C ✓
5	export-processor-consume		{"calculations": [{"op": "COUNT"}, {"filters": [{"column": "name", "op": "EQUAL", "value": "export-processor-consume"}]}]}	C ✓
6	What's the P99 memory usage of my nodes?		{"calculations": [{"column": "k8s.node.memory.usage", "op": "P99"}, {"time_range": 7200}]}]	I ✓
7	WHERE the baseline and selection are most different.		[{"column": "baseline", "order": "descending"}, {"column": "selection", "order": "descending"}]	I ✓
8	latency distribution by status code		{"breakdowns": [{"http.status_code": true}, {"calculations": [{"column": "duration_ms", "op": "HIGHEST"}]}]}	C ✓
9	show me latest traces		{"calculations": [{"op": "COUNT"}, {"orders": [{"op": "COUNT", "order": "descending"}]}]}	I ✓
10	Show me CPU utilization broken down by node		{"breakdowns": [{"k8s.node.name": true}, {"calculations": [{"column": "k8s.node.cpu.utilization", "op": "SUM"}]}]}	C ✓

critique openai/gpt-4-turbo

5/27/2024, 12:12:32 PM— 54 sec

accuracy

0.58DATASET
queries — 50 samplesPLAN
system_message → prompt_with_schema → generateSCORER
critique_scorer

Samples Info Logging JSON

Scores: All Sort: sample asc Close All Open All

	Input	Target	Answer	Score
1	show respond time heatmap		{"calculations": [{"column": "duration_ms", "op": "HE ATMAP"}]}	C
2	i want to search for traces which have graphql.operation.name = GenerateBartFromFileUpload and don't have a span where workflowType=processFileUploadWorkflow		{"calculations": [{"op": "COUNT"}], "filters": [{"column": "graphql.operation.na..."]}}	I
3	latest traces		{"calculations": [{"op": "COUNT"}], "orders": [{"op": "COUNT", "order": "descendi..."]}}	I

Messages Scoring Metadata

Input
latest traces

Target	Answer	Score
none	{"calculations": [{"op": "COUNT"}], "orders": [{"op": "COUNT", "order": "descending"}], "time_range": 3600}	I

Explanation

The query does not meet the needs of the NLQ for several reasons. The NLQ seeks a count of all pods starting with a specific name, hinting at a need for a wildcard or more inclusive name match that captures all relevant pods. However, the provided query specifies an exact start string for 'k8s.pod.name' which might not capture all relevant pods if the exact naming pattern varies more than initially followed. Additionally, while a COUNT operation and a time range of one week are correctly used, the filter should include a 'starts-with' operation to effectively capture all variants of the pod name starting with 'cron-no-medication'. This ensures it inclusively counts all relevant pods based on the name pattern described in the NLQ.

4	slow requests	{"breakdowns": ["http.route"], "calculations": [{"column": "duration_ms", "op": "H..."}]}	C
5	export-processor-consume	{"calculations": [{"op": "COUNT"}], "filters": [{"column": "name", "op": "=", "value..."}]}	C

Solvers

A Solver is a Python function that tasks a **TaskState** and transforms it in some useful fashion

TaskState (initialised from sample)

```
1 class TaskState:  
2     messages: list[ChatMessage]  
3     output: ModelOutput  
4     ...
```

Solver Function

```
1 async def solve(state: TaskState, generate: Generate) -> TaskState:  
2     # do something useful with state (prompt engineering,  
3     # generating model output, critique and regenerate, etc.)  
4     return state
```

Baseline Solvers

prompt_template()

```
1 async def solve(state: TaskState, generate: Generate) -> TaskState:  
2     prompt = state.user_prompt  
3     prompt.text = prompt_template.format(prompt=prompt.text, **params)  
4     return state
```

Modifies the existing prompt by passing it through a template

generate()

```
1 async def solve(state: TaskState, generate: Generate) -> TaskState:  
2     return await generate(state)
```

Calls the model, appends the assistant message, and updates the model output

Solver: `multiple_choice()`

Prompt with several choices (optionally shuffled)

```
1  async def solve(state: TaskState, generate: Generate) -> TaskState:
2
3      # build choices str and key
4      choices_str, choices_key = make_choices(choices=state.choices)
5
6      # re-write prompt with A,B,C,... choices
7      state.user_prompt.text = template.format(
8          question=state.user_prompt.text,
9          choices=choices_str,
10         )
11
12     # generate
13     state = await generate(state, temperature=0.0, max_tokens=1)
14
15     # map the output back to the right index and return
16     state.output.completion = choices_key[state.output.completion]
17
18     return state
```

Solver: self_critique()

Critique the generated response (possibly with another model), then re-generate in response to the critique.

```
1  async def solve(state: TaskState, generate: Generate) -> TaskState:
2
3      critique = await model.generate(
4          critique_template.format(
5              question=state.input_text,
6              completion=state.output.completion,
7          )
8      )
9
10     state.messages.append(ChatMessageUser(
11         content=completion_template.format(
12             question=state.input_text,
13             completion=state.output.completion,
14             critique=critique.completion,
15         ),
16     ))
17
18     return await generate(state)
```

Composition

Eval development frequently involves creating custom solvers and scorers. If made available in a Python package these can re-used across many evals

Some jailbreaking solvers from an internal **sheppard** package:

<code>encode()</code>	Message obfuscation jailbreak
<code>pap_jailbreak()</code>	Persuasion Adversarial Prompt (PAP)
<code>payload_splitting()</code>	PARROT jailbreak
<code>cr_jailbreak()</code>	Content reinforcement

Composition

Using **sheppard** to provide jailbreaks for a security eval:

```
1 from inspect_ai import Task, eval, task
2 from inspect_ai.scorer import model_graded_fact
3 from inspect_ai.solver import generate, system_message
4
5 from sheppard import pap_jailbreak
6
7 @task
8 def security_guide():
9     return Task(
10         dataset=example_dataset("security_guide"),
11         plan=[
12             system_message("system.txt"),
13             pap_jailbreak(),
14             generate()
15         ],
16         scorer=model_graded_fact(model="openai/gpt-4"),
17     )
```

Tool Use

TaskState also includes tools:

```
1 class TaskState:  
2     messages: list[ChatMessage]  
3     tools: list[ToolDef]  
4     tool_choice: ToolChoice  
5     output: ModelOutput  
6     ...
```

use_tools() makes tools available to generate():

```
1 return Task(  
2     dataset=example_dataset("biology_qa"),  
3     plan=[  
4         use_tools(web_search()),  
5         generate()  
6     ],  
7     scorer=model_graded_qa(template=GRADER_TEMPLATE),  
8 )
```

Agents and Tools

- Many permutations of agents and tool use are possible
- Bespoke agent logic inside a solver (swapping various tools in and out)
- Bridges to various agent libraries are as solvers (e.g. `langchain_agent()`,
`langroid_agent()`, etc.)

Agent: Capture the Flag

Cybersecurity eval using hand-rolled agent loop (custom agents and agent frameworks can both be embedded in solvers)

```
1 Plan(  
2     steps=[  
3         init_challenge(),  
4         use_tools([  
5             command_exec(), create_file(),  
6             decompile(), disassemble(),  
7             check_flag(),  
8         ]),  
9         system_message("prompts/system.txt"),  
10        initial_user_message(),  
11        generate(),  
12        check_for_flag_or_continue()  
13    ],  
14    cleanup=exit_challenge()  
15 )
```

Agent: LangChain

Convert any LangChain agent into a Solver

```
1 @solver
2 def wikipedia_search() -> Solver:
3
4     tavily_api = TavilySearchAPIWrapper()
5     tools = ([TavilySearchResults(api_wrapper=tavily_api)] +
6             load_tools(["wikipedia"]))
7
8     async def agent(llm: BaseChatModel, input: dict[str, Any]):
9         tools_agent = create_openai_tools_agent(llm, tools, prompt)
10        agent_executor = AgentExecutor.from_agent_and_tools(
11            agent=tools_agent,
12            tools=tools
13        )
14        result = await agent_executor.ainvoke(input)
15        return result["output"]
16
17    return langchain_solver(agent)
```

inspect-llm-workshop

INSPECT

CONFIGURATION (.ENV)

- Model Logging
- Model: anthropic (6)
- Model: claude-3-sonnet-20240229
- Connections: default, Retries: default, Timeout: default

TASK wikipedia

- Options Task Args
- Limit: default, Epochs: default
- Max Tokens: default, Temperature: default
- Top P: default, Top K: default

TASKS

- benchmarks
 - arc.py
 - arc_challenge
 - arc_easy
 - gpqa.py
 - gsm8k.py
 - mathematics.py
 - mmlu.py
- honeycomb
 - queries.py
 - critique
 - validate
- langchain
 - wikipedia.py
 - wikipedia

```

13  from langchain_core.language_models import BaseChatModel
14
15  from inspect_ai import Task, task
16  from inspect_ai.dataset import json_dataset
17  from inspect_ai.scorer import model_graded_fact
18  from inspect_ai.solver import Solver, solver
19
20  @task
21  def wikipedia() -> Task:
22      return Task(
23          dataset=json_dataset("wikipedia.jsonl"),
24          plan=wikipedia_search(),
25          scorer=model_graded_fact(),
26      )
27
28  @solver
29  def wikipedia_search(
30      max_iterations: int | None = 15,
31      max_execution_time: float | None = None
32  ) -> Solver:
33      # standard prompt for functions agent
34      prompt = hub.pull("hchase17/openai-tools-agent")
35
36      # tavily and wikipedia tools
37      tavily_api = TavilySearchAPIWrapper() # type: ignore
38      tools = [TavilySearchResults(api_wrapper=tavily_api)]

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

● (.venv) jjallaire@JJ's-MacBook-Pro inspect-llm-workshop % cd /Users/jjallaire/ukinstitute/inspect-llm-workshop
● (.venv) jjallaire@JJ's-MacBook-Pro inspect-llm-workshop % inspect eval langchain/wikipedia.py@wikipedia
  wikipedia (3 samples)

```

```

total time: 0:00:41
anthropic/clause-3-sonnet-20240229 11,514 tokens [9,400 + 2,114]

accuracy: 0.667 bootstrap_std: 0.268

Log: ./logs/2024-05-27T10-39-14_wikipedia_gWQ6UqJRde64cUhCdKufgr.json

```

Inspect View		Inspect View		Inspect View	
		2024-05-27T10-39-14_wikipedia_gWQ6UqJRde64cUhCdKufgr.json			
wikipedia anthropic/clause-3-sonnet-20240229		5/27/2024, 10:39:14 AM— 41 sec		accuracy bootstrap_std	
DATASET	wikipedia — 3 samples	PLAN	wikipedia_search	SCORER	model_graded_fact
Samples	Info	Logging	JSON	Scores:	All Sort: sample asc
1	List the ten episode titles from the sixth season of "Game of Thrones" in broadcast order.	The Red Woman, Home,...	From the summary, I can see...	C	
2	What's the difference between tennis and pickleball?	While they are similar sports,...	The key differences between...	C	
3	Which types of fish contain the lowest levels of mercury?	The following types of...	Based on the informati...	I	



inspect-llm-workshop

The screenshot shows a Jupyter Notebook interface with several panes:

- Left Sidebar:** Includes icons for file operations, search, connections, tasks, and help.
- Top Bar:** Shows the title "inspect-llm-workshop".
- Code Editor:** Displays Python code for interacting with Wikipedia using LangChain and Inspect AI. The code defines a task for fetching episode titles from Game of Thrones season 6 and a solver for executing the task using a Tavily API wrapper.
- Output:** Shows the command to run the code and the resulting output. The output includes metrics like total time (0:00:41), tokens (11,514), accuracy (0.667), and bootstrap std (0.268). It also shows the log path: ./logs/2024-05-27T10-39-14_wikipedia_gWQ6UqJRde64cUhCdKufgr.json.
- Inspect View:** A detailed view of the inspection process. It shows a table with rows for user queries, system responses, and AI-generated answers. The first row is highlighted with a green checkmark. Below the table, there are tabs for "Messages", "Scoring", and "Metadata".
- Bottom Right:** A sidebar with icons for "zsh", "Inspect ...", and "Quarto P...".

inspect-llm-workshop

```

INSPECT ... wikipedia.py ...
langchain > wikipedia.py > ...
13 from langchain_core.language_models import BaseChatModel
14
15 from inspect_ai import Task, task
16 from inspect_ai.dataset import json_dataset
17 from inspect_ai.scorer import model_graded_fact
18 from inspect_ai.solver import Solver, solver
19
@task
def wikipedia() -> Task:
    return Task(
        dataset=json_dataset("wikipedia.jsonl"),
        plan=wikipedia_search(),
        scorer=model_graded_fact(),
    )
@solver
def wikipedia_search(
    max_iterations: int | None = 15,
    max_execution_time: float | None = None
) -> Solver:
    # standard prompt for functions agent
    prompt = hub.pull("hwchase17/openai-tools-agent")
    # tawily and wikipedia tools
    tawily_api = TawilySearchAPIWrapper() # type: ignore
    tools = [TawilySearchResults(api_wrapper=tawily_api)]

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

- (.venv) jjallaire@JJ's-MacBook-Pro inspect-llm-workshop % cd /Users/jjallaire/ukinstitute/inspect-llm-workshop
- (.venv) jjallaire@JJ's-MacBook-Pro inspect-llm-workshop % inspect eval langchain/wikipedia.py@wikipedia wikipedia (3 samples)

```

total time: 0:00:41
anthropic/clause-3-sonnet-20240229 11,514 tokens [9,400 + 2,114]
accuracy: 0.667 bootstrap_std: 0.268
Log: ./logs/2024-05-27T10-39-14_wikipedia_gWQ6UqJRde64cUhCdKufgr.json

```

anthropic/clause-3-sonnet-20240229 dataset: wikipedia scorer: model_graded_fact

INSPECT View X

Inspect View 2024-05-27T10-39-14_wikipedia_gWQ6UqJRde64cUhCdKufgr.json

3 Which types of fish contain the lowest levels of mercury? The following types of... Based on the informati...

Messages Scoring Metadata

Input
Which types of fish contain the lowest levels of mercury?

Target
The following types of fish contain low levels of mercury:
salmon, flounder, Atlantic mackerel, anchovies, pollock, catfish, and shellfish (e.g., clams, scallops, mussels).

Answer
Based on the information from Wikipedia, some key points about mercury levels in fish:

- Fish higher up the food chain like tuna, swordfish, shark tend to have higher mercury levels due to bioaccumulation
- Long-lived, predatory fish concentrate more mercury in their bodies over time
- Small fish that feed on plankton and are lower on the food chain tend to have lower mercury levels

Therefore, to minimize mercury exposure from fish, it is recommended to eat smaller fish species that are

zsh Inspect ... Quarto P...

main* 0 0 0 Quarto: 1.4.553

inspect-llm-workshop

INSPECT

CONFIGURATION (.ENV)

Model Logging

Model: anthropic (6)

Model: claude-3-sonnet-20240229

Connections: default, Retries: default, Timeout: default

TASK wikipedia

Options Task Args

Limit: default, Epochs: default

Max Tokens: default, Temperature: default

Top P: default, Top K: default

TASKS

- benchmarks**
 - arc.py (arc_challenge)
 - arc_easy
 - gpqa.py
 - gsm8k.py
 - mathematics.py
 - mmlu.py
- honeycomb**
 - queries.py
 - critique
 - validate
- langchain**
 - wikipedia.py (1)
 - wikipedia

wikipedia.py

```

13  from langchain_core.language_models import BaseChatModel
14
15  from inspect_ai import Task, task
16  from inspect_ai.dataset import json_dataset
17  from inspect_ai.scorer import model_graded_fact
18  from inspect_ai.solver import Solver, solver
19
20  #> Debug Task | > Run Task
21  @task
22  def wikipedia() -> Task:
23      return Task(
24          dataset=json_dataset("wikipedia.jsonl"),
25          plan=wikipedia_search(),
26          scorer=model_graded_fact(),
27      )
28
29  @solver
30  def wikipedia_search(
31      max_iterations: int | None = 15,
32      max_execution_time: float | None = None
33  ) -> Solver:
34      # standard prompt for functions agent
35      prompt = hub.pull("hchase17/openai-tools-agent")
36
37      # tavily and wikipedia tools
38      tavily_api = TavilySearchAPIWrapper() # type: ignore
      tools = [TavilySearchResults(api_wrapper=tavily_api)]

```

PROBLEMS

- (.venv) jjallaire@JJ's-MacBook-Pro inspect-llm-workshop % cd /Users/jjallaire/ukinstitute/inspect-llm-workshop
- (.venv) jjallaire@JJ's-MacBook-Pro inspect-llm-workshop % inspect eval langchain/wikipedia.py@wikipedia

wikipedia (3 samples)

```

total time: 0:00:41
anthropic/claude-3-sonnet-20240229 11,514 tokens [9,400 + 2,114]
accuracy: 0.667 bootstrap_std: 0.268
Log: ./logs/2024-05-27T10-39-14_wikipedia_gWQ6UqJRde64cUhCdKufgr.json

```

Inspect View

2024-05-27T10-39-14_wikipedia_gWQ6UqJRde64cUhCdKufgr.json

Explanation

To assess the submitted answer against the expert answer, I will follow these steps:

1. List out the types of fish mentioned as low-mercury in the expert answer.
2. Check if the submitted answer includes all those fish types.
3. Check if the submitted answer includes any additional low-mercury fish types not present in the expert answer.
4. Based on the above points, determine if the submitted answer fully contains the content from the expert answer.

Step 1: Types of low-mercury fish mentioned in the expert answer:

- salmon
- flounder
- Atlantic mackerel
- anchovies
- pollock
- catfish
- shellfish (clams, scallops, mussels)

Step 2: Checking if the submitted answer includes all those fish types:

- salmon: Yes
- flounder: No
- Atlantic mackerel: No
- anchovies: Yes

TERMINAL

PORTS

zsh

Inspect ...

Quarto P...

29

Scoring

- Conventional pattern matching / templated answer based scoring built in.
- Model graded scorer built-in (often heavily customized)
- Scorers also pluggable (i.e. provided from other packages). We expect lots of innovation in model graded scoring!
- Offline / human scoring workflow is supported.
- Plan to build tools to help rigorously evaluate model graded scorers against human baselines.

INSPECT ...

CONFIGURATION (.ENV)

- Model Logging
- Model openai (10)
- gpt-4-turbo
- Connections default Retries default Timeout default

TASK math ⚡ ▶

- Options Task Args
- Limit 10 Epochs default
- Max Tokens default Temperature default
- Top P default Top K default

TASKS

- benchmarks
 - arc.py
 - arc_challenge
 - arc_easy
 - gpqa.py
 - gsm8k.py
 - mathematics.py
 - math
 - mmlu.py
- honeycomb
 - queries.py
 - critique
 - validate
- langchain
- wikipedia.py

mathematics.py x

```

benchmarks > mathematics.py > ...
  ⌂ Debug Task | ⌂ Run Task
38 @task
39 def math(shuffle=True):
40     return Task(
41         dataset=csv_dataset(
42             csv_file="datasets/math_test.csv",
43             sample_fields=FieldSpec(input="Question",
44             target="Answer"),
45             shuffle=shuffle,
46         ),
47         plan=[
48             prompt_template(PROMPT_TEMPLATE),
49             generate(),
50         ],
51         scorer=expression_equivalence(),
52         config=GenerateConfig(temperature=0.5),
53     )
54
55 @scorer(metrics=[accuracy(), bootstrap_std()])
56 def expression_equivalence():
57     async def score(state: TaskState, target: Target):
58         # extract answer
59         match = re.search(AnswerPattern.LINE, state.
60                           output.completion)
61         if match:
62             # ask the model to judge equivalence

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

● (.venv) jjallaire@JJ's-MacBook-Pro inspect-llm-workshop % inspect eval benchmarks/mathematics.py@math --limit 10
math (10 samples) —
temperature: 0.5

total time: 0:00:27
openai/gpt-4-turbo 8,775 tokens [4,723 + 4,052]

accuracy: 0.8 bootstrap_std: 0.123
Log: ./logs/2024-05-27T10-52-10_math_4RW2MmctfNi9bBoPBjvKQo.json

```

Inspect View x

math openai/gpt-4-turbo
5/27/2024, 10:52:10 AM – 27 sec

DATASET math_test – 10 samples PLAN prompt_template → generate

HYPERPARAMETERS temperature: 0.5

Samples Info Logging JSON Scores: All Sort: sample asc Open All

	Input	Target	Answer	Score
1	A circle of radius \$2\$ is inscribed in a semicircle, as shown. The area inside the semicircle but outside...	$\frac{1}{2}$	$\frac{1}{2}$	C
2	Let f be a function taking the positive integers to the positive integers, such that...	18	2007	I
3	Mary has \$6\$ identical basil plants, and three different window sills she can put them on. How...	28	28	C
4	Compute: $(3^2)(2^4)(37)(5^3)$	666000	666000	C
5	There are 30 men and 40 women in the Town Library Club. They wish to form a 7-person steering...	371,043,!	371043400	C

zsh Inspect ... Quarto P...

main* ⌂ 0 ⌂ 0 ⌂ 0 ⌂ 0 Quarto: 1.4.553 Ln 53, Col 1 Spaces: 2 UTF-8 LF Python 3.11.6 ('.venv': venv) Prettier 31

Scorer: expression_equivalence()

```
1 @scorer(metrics=[accuracy(), bootstrap_std()])
2 def expression_equivalance():
3     async def score(state: TaskState, target: Target):
4
5         # extract answer
6         match = re.search(AnswerPattern.LINE, state.output.completion)
7
8         # ask the model to judge equivalence
9         answer = match.group(1)
10        prompt = EQUIVALENCE_TEMPLATE %
11            {"expression1": target.text, "expression2": answer}
12        )
13        result = await get_model().generate(prompt)
14
15        # return the score
16        correct = result.completion.lower() == "yes"
17        return Score(
18            value=CORRECT if correct else INCORRECT,
19            answer=answer,
20            explanation=state.output.completion,
21        )
22
23    return score
```

Logging

- Capture all context required to debug, analyse, and reproduce evaluations
- Python API for computing on log file contents
- Log viewer for interactive exploration of eval results

EvalLog

status	str	Status of evaluation
eval	EvalSpec	Top level eval details including task, model, creation time, etc.
plan	EvalPlan	List of solvers and model generation config used for the eval.
samples	list[EvalSample]	Each sample evaluated, including its input, output, target, and score.
results	EvalResults	Aggregated scorer results
stats	EvalStats	Model token usage stats
logging	list[LoggingMessage]	Logging messages (e.g. from <code>log.info()</code> , <code>log.debug()</code> , etc.)
error	EvalError	Error information

Log Viewer: Samples

gpqa_diamond	openai/gpt-4	accuracy 0.335	bootstrap_std 0.017
4/28/2024, 10:18:00 AM— 10 min 59 sec			
DATASET gpqa_diamond — 198 x 4 samples	PLAN multiple_choice	SCORER answer	
HYPERPARAMETERS temperature: 0.5			
Samples	Info	Logging	JSON
	Epochs: 2	Scores: All	Sort: epoch asc
			Open All
Input	Target	Answer	Score
EPOCH 2			
1 Two quantum states with energies E1 and E2 have a lifetime of 10^{-9} sec and 10^{-8} sec, respectively. We want to clearly distinguish these two energy levels. Which one ...	A	A	C ▾
2 trans-cinnamaldehyde was treated with methylmagnesium bromide, forming product 1. 1 was treated with pyridinium chlorochromate, forming...	A	The question seems to contain a mistake. Product 3 is formed...	I ▾
3 A spin-half particle is in a linear superposition $0.5 \uparrow\rangle + \sqrt{3}/2 \downarrow\rangle$ of its spin-up and spin-down states. If $ \uparrow\rangle$ and...	A	The expectation value of an operator O in a state $ \psi\rangle$ is given by...	I ▾
4 In a parallel universe where a magnet can have an isolated North or South pole, Maxwell's equations look different. But, specifically, which of those equations are different?	A	C	I ▾
5 Calculate the eigenvector of a quantum mechanical operator \vec{P} for a muon along an arbitrary direction \vec{n} lying in the x-z plane corresponding to the...	A	A	C ▾

Log Viewer: Messages

1 What's the difference between tennis and pickleball? While they are similar sports, tennis and pickleball, while similar in some...  

Messages Scoring Metadata

 **system**
You are a helpful assistant

 **user**
What's the difference between tennis and pickleball?

 **assistant**
 wikipedia(__arg1: Pickleball)

Page: Pickleball
Summary: Pickleball is a racket or paddle sport in which two players (singles) or four players (doubles) hit a perforated, hollow plastic ball with paddles over a 34-inch-high (0.86 m) net until one side is unable to return the ball or commits a rule infraction. Pickleball is played indoors and outdoors. It was invented in 1965 as a children's backyard game in the United States, on Bainbridge Island in Washington state. In 2022, pickleball was named the official state sport of Washington.

 **assistant**
 wikipedia(__arg1: Tennis)

Page: Tennis
Summary: Tennis is a racket sport that is played either individually against a single opponent (singles) or between two teams of two players each (doubles). Each player uses a tennis racket that is strung with cord to strike a hollow rubber ball covered with felt over or around a net and into the opponent's court. The object of the game is to manoeuvre the ball in such a way that the opponent is not able to play a valid return. The player who is unable to return the ball validly will not gain a point, while the opposite player will.

 **assistant**
Tennis and pickleball, while similar in some ways, have several fundamental differences.

In tennis:

- It's a racket sport that can be played individually against a single opponent (singles) or between two teams of two players each (doubles).

Log Viewer: Scoring

The universe is filled with the Cosmic Microwave Background. Consider the annihilation of high energy \gamma-rays with a photon from the CMB Radiation into...

3 A A C ^

Messages Scoring

Input
The universe is filled with the Cosmic Microwave Background. Consider the annihilation of high energy \gamma-rays with a photon from the CMB Radiation into electron-positron, i.e. $\gamma\gamma \rightarrow e^+e^-$. From what energy \gamma-rays would have their lifetimes in the universe limited by this process? Knowing that the average photon energy of the CMB is 10^{-3} eV.

A) 2.6×10^5 GeV
B) 1.8×10^5 GeV
C) 3.9×10^5 GeV
D) 9.5×10^4 GeV

Target	Answer	Score
A	A	C

Explanation
The process of photon-photon pair production (or annihilation) can only occur if the energy of the incoming gamma-ray photon is above a certain threshold. This threshold is given by the rest mass energy of the electron-positron pair, which is about 1 MeV or 10^6 eV. However, the energy of the incoming photon must also be significantly higher than the energy of the CMB photon it is interacting with.

The energy of the CMB photon is given as 10^{-3} eV. The energy of the gamma-ray photon must be such that, in the rest frame of the CMB photon, the gamma-ray photon has an energy greater than 1 MeV. This requires a gamma-ray energy of approximately 2×10^{11} eV or 2×10^5 GeV in the lab frame.

Looking at the answer choices, the closest to this value is 2.6×10^5 GeV.

ANSWER: A

Models

Provider	Model Name	Docs
OpenAI	openai/gpt-3.5-turbo	OpenAI Models
Anthropic	anthropic/clause-3-sonnet-20240229	Anthropic Models
Google	google/gemini-1.0-pro	Google Models
Mistral	mistral/mistral-large-latest	Mistral Models
Hugging Face	hf/openai-community/gpt2	Hugging Face Models
Ollama	ollama/llama3	Ollama Models
TogetherAI	together/lmsys/vicuna-13b-v1.5	TogetherAI Models
AWS Bedrock	bedrock/meta.llama2-70b-chat-v1	AWS Bedrock Models
Azure AI	azureai/azure-deployment-name	Azure AI Models
Cloudflare	cf/meta/llama-2-7b-chat-fp16	Cloudflare Models

Interface with any other model by creating a custom model provider...

Workflow

- Lots of interactive exploration occurs during eval development, so critical to have good support for iterating in a Notebook / REPL
- Eventually though, evals need to end up in a form that enables reproducibly running them in an eval suite
- Need to support a continuum of workflows that transition well into each other
- Provide good tooling in Jupyter and VS Code for entire spectrum

localhost

File Edit View Run Kernel Tabs Settings Help

Launcher queries.ipynb Python 3 (ipykernel)

Validate Task

Now we'll put all of this together to create an evaluation task:

```
[4]: from inspect_ai import eval, task, Task
from inspect_ai.solver import system_message, generate

@task
def validate():
    return Task(
        dataset=dataset,
        plan=[
            system_message("Honeycomb AI suggests queries based on user input."),
            prompt_with_schema(),
            generate()
        ],
        scorer=validate_scorer()
    )
```

We can run the task using Inspect's `eval()` function (limiting to 100 samples):

```
[*]: eval(validate, model="openai/gpt-4-turbo", limit=100)
```

validate (100 samples) 69% 0:00:14 openai/gpt-4-turbo dataset: queries scorer: validate_scorer
openai/gpt-4-turbo: 10/10 HTTP rate limits: 0

Critique Task

Now we'll create a critique task. For this we'll need a LLM-based scorer that uses a critique template to prompt for whether the generated query is "good" or "bad":

```
[ ]: import json
from inspect_ai.model import get_model

@scorer(metrics=[accuracy()])
def critique_scorer(model = "anthropic/clause-3-opus-20240229"):

    async def score(state, target):

        # build the critic prompt
        query = state.output.completion.strip()
        critic_prompt = resource("critique.txt").replace(
            "{{prompt}}", state.user_prompt.text
```

Interactive Exploration

Ad-hoc exploration of an eval in a Notebook/REPL

```
1 params = {  
2     "system": [ "devops.txt", "researcher.txt" ],  
3     "grader": [ "hacker.txt", "expert.txt" ],  
4     "grader_model": [ "openai/gpt-4", "google/gemini-1.0-pro" ]  
5 }  
6 params = list(product(*(params[name] for name in params)))  
7  
8 tasks = [Task(  
9     dataset=json_dataset("security_guide.jsonl"),  
10    plan=[system_message(system), generate()],  
11    scorer=model_graded_fact(template=grader, model=grader_model)  
12 ) for system, grader, grader_model in params]  
13  
14 logs = eval(tasks, model = "mistral/mistral-large-latest")  
15 plot_results(logs)
```

Task Parameters

Formalise variation with a parameterised `@task` function:

```
1  @task
2  def security_guide(system="devops.txt", grader="expert.txt"):
3      return Task(
4          dataset = json_dataset("security_guide.jsonl"),
5          plan=[system_message(system), generate()],
6          scorer=model_graded_fact(template=grader, model="openai/gpt-4")
7      )
8
9  params = {
10     "system": [ "devops.txt", "researcher.txt" ],
11     "grader": [ "hacker.txt", "expert.txt" ]
12 }
13 params = list(product(*(params[name] for name in params)))
14
15 eval([security_guide(system,grader) for system, grader in params],
16       model = "mistral/mistral-large-latest")
```

Task Parameters

@task functions are registered and addressable by external driver programs
(step one in development => production)

```
1 @task
2 def security_guide(system="devops.txt", grader="expert.txt"):
3     return Task(
4         dataset = json_dataset("security_guide.jsonl"),
5         plan=[system_message(system), generate()],
6         scorer=model_graded_fact(
7             template=grader,
8             model="openai/gpt-4"
9         )
10    )
```

Now we can vary the parameters externally:

```
1 $ inspect eval security_guide.py -T system=devops.txt
2 $ inspect eval security_guide.py -T grader=hacker.txt
```

Same workflow available for tasks in a notebook:

```
1 $ inspect eval security_guide.ipynb -T system=devops.txt
2 $ inspect eval security_guide.ipynb -T grader=hacker.txt
```

Task Variants

We may discover that we *always* want to vary a parameter when running a full evaluation suite:

```
1 def security_guide(system, grader="expert.txt"):
2     return Task(
3         dataset = json_dataset("security_guide.jsonl"),
4         plan=[system_message(system), generate()],
5         scorer=model_graded_fact(template=grader, model="openai/gpt-4")
6     )
7
8 @task
9 def devops():
10    return security_guide("devops.txt")
11
12 @task
13 def researcher():
14    return security_guide("researcher.txt")
```

Invoke by task name

```
1 $ inspect eval security_guide.py@devops
2 $ inspect eval security_guide.py@researcher
```

Eval Suites

We want to allow for arbitrary source code organisation but still be able to discover and enumerate tasks for a suite

```
1 security/  
2 jeopardy/  
3     import.py  
4     analyze.py  
5     task.py  
6 attack_defense/  
7     import.py  
8     analyze.py  
9     task.py
```

```
1 list_tasks("security")  
2  
3 jeopardy/task.py@crypto  
4 jeopardy/task.py@decompile  
5 jeopardy/task.py@packet  
6 jeopardy/task.py@heap_trouble  
7 attack_defense/task.py@saar  
8 attack_defense/task.py@bank  
9 attack_defense/task.py@voting
```

Run them all

```
1 eval(list_tasks("security"), model="mistral/mistral-large-latest")
```

Resiliency

The production version would look more like this:

```
1 # setup log context
2 os.environ["INSPECT_LOG_DIR"] = "./security-suite_04-07-2024"
3
4 # run the eval suite
5 tasks = list_tasks("security")
6 eval(tasks, model="mistral/mistral-large-latest")
7
8 # ...later, in another process that also has INSPECT_LOG_DIR
9 error_logs = list_eval_logs(status == "error")
10 eval_retry(error_logs)
```

Somewhat oversimplified, as we'd also want to enhance the logic around analysing the cause of errors and adopting optimal recovery strategies

Provenance

If you run an eval from a Git repository, you should be able to reproduce the eval with only its log file as context

```
1 # read the log and extract the origin and commit
2 log = read_eval_log("security-log.json")
3 origin = log.spec.revision.origin
4 commit = log.spec.revision.commit
5
6 # clone the repo, checkout the commit, install deps, and run
7 run(["git", "clone", revision.origin, "eval-dir"])
8 with chdir("eval-dir"):
9     run(["git", "checkout", revision.commit])
10    run(["pip", "install", "-r", "requirements.txt"])
11    eval(log)
```

Learning More

- Docs: https://ukgovernmentbeis.github.io/inspect_ai
- GitHub: https://github.com/ukgovernmentbeis/inspect_ai
- Slides/Code: <https://github.com/jjallaire/inspect-llm-workshop>

Questions?