plotting

April 15, 2025

[2]:
```
# Standard library imports
import collections
import contextlib
import copy
import dataclasses as dc
import datetime
import enum
import functools
import gc
import hashlib
import itertools
import json
import os
import pathlib
import re
import sys
import time
from typing import Any

# Third-party imports
import datasets
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import polars as pl
import rich
import rich.console
import rich.table
import tqdm
from ast import literal_eval
from IPython.display import display, Markdown
import more_itertools as mit
import itertools as it

os.environ["OPENINSTRUCT_PARSE_LATEX_BACKEND"] = "lark"
```

```
sys.path.append("/home/mila/g/gagnonju/marglicot/with_open-instruct/
 ↪open-instruct")
from open_instruct.math_utils import (
    last_boxed_only_string,
    remove_boxed,
    get_unnormalized_answer,
    normalize_final_answer,
    is_equiv,
    hendrycks_is_equiv
)


def md_print(text):
    display(Markdown(text))


# Make Pandas display better.
pd.set_option('display.max_colwidth', None)
pd.set_option('display.width', 1000)
pl.Config(fmt_str_lengths=500, tbl_width_chars=10000, tbl_rows=100,␣
 ↪tbl_cell_alignment="LEFT")



class Mode(enum.Enum):
    gsm8k = "gsm8k"
    math = "math"

class LearningType(enum.Enum):
    sft = "sft"
    rejection = "rejection"
    zero_shot = "zero_shot"
    few_shot = "few_shot"
```

```
/home/mila/g/gagnonju/marglicot/light_eval_tests/.venv/lib/python3.11/site-
packages/tqdm/auto.py:21: TqdmWarning: IProgress not found. Please update
jupyter and ipywidgets. See
https://ipywidgets.readthedocs.io/en/stable/user_install.html
  from .autonotebook import tqdm as notebook_tqdm
```

0.1 Extract the paths of all of the per epoch saves

We need all the save paths. - We find all of the saves by looking for all of the safetensors files - There is one save per epoch, & the save directories are …/model/checkpoint_name.safetensors, so we call .parent.parent to get the run directory.

[3]:
```
SAVE_DIRECTORY = "~/scratch/marglicot_saves/sft_saves"
CHECKPOINT_GLOB_PATTERN = "**/*.safetensors"

paths_saves_per_checkpoint = []
for x in pathlib.Path(SAVE_DIRECTORY).expanduser().
 ↪glob(CHECKPOINT_GLOB_PATTERN):
    paths_saves_per_checkpoint.append(x.parent.parent)
display(Markdown(f"We found {len(paths_saves_per_checkpoint)} saves."))
```

We found 388 saves.

1 Centralize paths per run, sorting by creation time.

We need to know which runs exist. We need to know: - When they were created. - How many epochs (checkpoints) they have.

We will then need to plot the accuracy per epoch for a selection of runs.

[4]:
```
by_run = collections.defaultdict(list)
for path in paths_saves_per_checkpoint:
    by_run[path.parent].append(path)

md_print(f"We found **{len(by_run)}** runs.")

high_level_info = []
for run, paths in by_run.items():
    high_level_info.append({
        "run": run,
        "num_paths": len(paths),
        "creation_time": datetime.datetime.fromtimestamp(run.stat().st_ctime).
 ↪strftime('%Y-%m-%d %H:%M:%S'),
        "creation_time_raw": run.stat().st_ctime
    })

high_level_info = pd.DataFrame(high_level_info)
high_level_info = high_level_info.sort_values('creation_time_raw',␣
 ↪ascending=True)


md_print("## >> Runs to **creation date** and **number of checkpoints**:")
print(high_level_info)
```

We found **29** runs.

1.1 » Runs to creation date and number of checkpoints:

```
                                   run  num_paths        creation_time
creation_time_raw
```

```
0                 /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_001/
ao_gsm8k_smollm2_1.7B-2025-03-10_01-28-43          2  2025-03-10 21:55:46
1.741658e+09
1                /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_001/c
ot_gsm8k_smollm2_1.7B-2025-03-10_01-28-46          7  2025-03-10 21:55:46
1.741658e+09
2                  /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_001
/ao_math_smollm2_1.7B-2025-03-10_01-28-46          2  2025-03-10 21:55:46
1.741658e+09
3                 /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_001/
cot_math_smollm2_1.7B-2025-03-10_01-46-43          7  2025-03-10 21:55:46
1.741658e+09
4                 /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_001/
cot_math_smollm2_1.7B-2025-03-10_01-49-58          7  2025-03-10 21:55:46
1.741658e+09
5                /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_00001
/ao_math_smollm2_1.7B-2025-03-10_01-31-43         10  2025-03-10 21:55:46
1.741658e+09
6              /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_00001/c
ot_gsm8k_smollm2_1.7B-2025-03-10_01-31-43          7  2025-03-10 21:55:46
1.741658e+09
7               /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_00001/
ao_gsm8k_smollm2_1.7B-2025-03-10_01-31-47         10  2025-03-10 21:55:46
1.741658e+09
8    /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_0005/checkpoints/
ao_gsm8k_smollm2_1.7B-2025-03-10_23-32-03         30  2025-03-15 23:27:19
1.742096e+09
9   /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/0_0005/checkpoints/c
ot_gsm8k_smollm2_1.7B-2025-03-10_23-32-14         30  2025-03-15 23:27:19
1.742096e+09
10                     /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/c
ot_gsm8k_smollm2_1.7B-2025-03-16_00-02-01         12  2025-03-16 01:58:54
1.742105e+09
11             /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k
_smollm2_1.7B_0_00001-2025-03-16_00-02-01         30  2025-03-16 04:52:01
1.742115e+09
12            /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_
smollm2_1.7B_0_000025-2025-03-16_02-39-20         30  2025-03-16 07:36:56
1.742125e+09
14           /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_s
mollm2_1.7B_0_00005_5-2025-03-16_19-34-20          5  2025-03-16 20:23:54
1.742171e+09
13         /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_smo
llm2_1.7B_0_000025_30-2025-03-16_19-25-20         21  2025-03-16 22:55:15
1.742180e+09
15         /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_smo
llm2_1.7B_0_000075_30-2025-03-16_19-40-20         20  2025-03-16 23:00:10
1.742180e+09
```

```
16          /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_sm
ollm2_1.7B_0_000075_5-2025-03-16_19-43-20         20  2025-03-16 23:00:27
1.742180e+09
17           /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_s
mollm2_1.7B_0_00025_5-2025-03-16_23-10-55          5  2025-03-17 00:00:49
1.742184e+09
18            /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_
smollm2_1.7B_0_0001_5-2025-03-16_23-10-55          5  2025-03-17 00:00:51
1.742184e+09
19          /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_sm
ollm2_1.7B_0_000075_5-2025-03-16_23-10-55         13  2025-03-17 01:19:41
1.742189e+09
20           /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_s
mollm2_1.7B_0_0001_15-2025-03-17_00-51-17          5  2025-03-17 01:41:06
1.742190e+09
22          /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_sm
ollm2_1.7B_0_000075_5-2025-03-17_01-26-01          5  2025-03-17 02:15:20
1.742192e+09
21         /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_gsm8k_smo
llm2_1.7B_0_000075_15-2025-03-17_01-26-01         15  2025-03-17 03:54:50
1.742198e+09
24           /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math_sm
ollm2_1.7B_0_00005_15-2025-04-06_02-03-21         15  2025-04-06 13:23:50
1.743960e+09
23            /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math_s
mollm2_1.7B_0_0005_15-2025-04-06_02-03-21         15  2025-04-06 13:29:53
1.743961e+09
25            /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math_s
mollm2_1.7B_0_0001_15-2025-04-06_02-03-21         15  2025-04-06 13:35:10
1.743961e+09
26              /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math
_smollm2_1.7B_0_01_15-2025-04-13_02-41-58         15  2025-04-13 13:48:41
1.744567e+09
27             /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math_
smollm2_1.7B_0_001_15-2025-04-13_02-41-58         15  2025-04-13 13:49:53
1.744567e+09
28             /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math_
smollm2_1.7B_0_005_15-2025-04-13_02-41-58         15  2025-04-13 13:52:51
1.744567e+09
```

2 Filter the runs that we care about

We only care about the runs that happened on the **2025-04-06**.

They are the latest runs at the time of writing.

We extract those.

[5]:
```
# Filter runs from 2025-04-06
# DATE_OF_INTEREST = "2025-03-17"
DATE_OF_INTEREST = "2025-04-06"
# DATE_OF_INTEREST = "2025-04-13"

filtered_runs = high_level_info[high_level_info['creation_time'].str.
 ↪contains(DATE_OF_INTEREST)]
display(Markdown("## >> Filtered runs to **creation date** and **number of␣
 ↪checkpoints**:"))
md_print(f"Found **{len(filtered_runs)}** runs from **{DATE_OF_INTEREST}**")
print(filtered_runs)

# Get the last date from all runs
last_date = high_level_info['creation_time'].str.split(' ').str[0].max()

# Check if the date of interest is the last date
if DATE_OF_INTEREST != last_date:
    md_print(f"<span style='color: red; font-weight: bold; font-size: 1.2em;'>␣
 ↪Warning: {DATE_OF_INTEREST} is not the last date. The last date is␣
 ↪{last_date}</span>")
else:
    md_print(f" {DATE_OF_INTEREST} is the last date.")
```

2.1 » Filtered runs to creation date and number of checkpoints:

Found **3** runs from **2025-04-06**

```
                                run  num_paths        creation_time
creation_time_raw
24  /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math_smollm2_1.7
B_0_00005_15-2025-04-06_02-03-21         15  2025-04-06 13:23:50
1.743960e+09
23   /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math_smollm2_1.
7B_0_0005_15-2025-04-06_02-03-21         15  2025-04-06 13:29:53
1.743961e+09
25   /home/mila/g/gagnonju/scratch/marglicot_saves/sft_saves/cot_math_smollm2_1.
7B_0_0001_15-2025-04-06_02-03-21         15  2025-04-06 13:35:10
1.743961e+09
```

Warning: 2025-04-06 is not the last date. The last date is 2025-04-13

3 Check if we ran inference on all of the filtered runs

To do that we need to look for all of the inference outputs in the all_eval_outputs_important directory.

We find the paths through pattern matching in the path names of the inference outputs directories.

- We find all eval outputs by finding all of the .parquet files in the all_eval_outputs_important directory.
- We extract the associated meta_info.json files, which contain the path to the associated run checkpoint directory.
- We make a join on the filtered runs and the extracted paths to get the runs that we did inference on.

[6]:
```
MODE = Mode.math
LEARNING_TYPE = LearningType.sft
EVAL_OUTPUTS_DIR = f"/home/mila/g/gagnonju/marglicot/light_eval_tests/
 ↪all_eval_outputs_important/{LEARNING_TYPE.value}_outputs_{MODE.value}/"
GLOB_PATTERN = "**/*.parquet"
MIN_MAX_NUM_EPOCHS = 15


paths_eval = [
    x.parent.parent for x in pathlib.Path(EVAL_OUTPUTS_DIR).glob(GLOB_PATTERN)
]

@dc.dataclass
class MetaInfo:
    inference_outputs_path: str | pathlib.Path
    save_path_from_json: str | pathlib.Path
    results_path: str | pathlib.Path
    meta_config: dict[str, Any]

############################################################
# Extract the info from the Eval Outputs
############################################################
ckpt_paths: list[MetaInfo] = []
for path_eval in paths_eval:
    # Replace "details" with "results" in the path

    details_idx = path_eval.parts.index("details")
    results_path = pathlib.Path(*(it.chain(path_eval.parts[:details_idx],␣
 ↪["results"], path_eval.parts[details_idx + 1:])))
    meta_info_path = results_path / "meta_info.json"

    with open(meta_info_path, "r") as f:
        meta_info = json.load(f)

    run_path = pathlib.Path(meta_info["save_path"]).resolve()
    # print(meta_info)
    ckpt_paths.append(MetaInfo(
        inference_outputs_path=path_eval.resolve(),
        save_path_from_json=run_path,
        results_path=results_path,
```

```
        meta_config=meta_info
    ))


assert len(ckpt_paths) == len(paths_eval)
# print("\n".join([str(x.save_path_from_json) for x in ckpt_paths]))

# For each run in filtered_runs, see for which ones we have inference outputs
# Create a mapping from run paths to their corresponding MetaInfo objects
run_path_to_meta = collections.defaultdict(lambda: collections.
 ↪defaultdict(list))
for meta in ckpt_paths:
    results_json = json.loads(pathlib.Path(mit.one(meta.results_path.glob("**/
 ↪results*.json"))).read_text())
    n_shots = int(mit.one(results_json["versions"]).rsplit("|")[-1])
    effective_few_shots = mit.one(results_json["summary_tasks"].
 ↪values())["effective_few_shots"]
    max_num_epochs = meta.meta_config["cfg"]["max_num_epochs"]
    if not MIN_MAX_NUM_EPOCHS or (max_num_epochs >= MIN_MAX_NUM_EPOCHS):
        run_path_to_meta[meta.save_path_from_json][n_shots].append(meta)

############################################################
# Extract the runs that have inference outputs
############################################################
filtered_runs_with_outputs = []
print()
for run in filtered_runs.itertuples():
    # Convert the run path to a Path object for comparison
    run_path = pathlib.Path(run.run).resolve()
    # print(run_path)
    if run_path in run_path_to_meta:
        filtered_runs_with_outputs.append(run_path_to_meta[run_path])
    else:
        print(f"Run {run_path} not found in run_path_to_meta")

md_print(f"Found **{len(filtered_runs_with_outputs)}** runs with inference␣
 ↪outputs out of **{len(filtered_runs)}** total filtered runs")
md_print(f"Found a total of **{len(list(it.chain.
 ↪from_iterable(filtered_runs_with_outputs)))}** checkpoints.")
for n_shot_to_run in filtered_runs_with_outputs:
    for n_shot, run in n_shot_to_run.items():
        max_num_epochs = mit.one(set(x.meta_config["cfg"]["max_num_epochs"] for␣
 ↪x in run))
        epochs = sorted(x.meta_config["epoch"] for x in run)
        missing = sorted(set(range(max_num_epochs)) - set(epochs))
```

```
        text_missing = "Missing epochs: " + ", ".join(str(x) for x in missing)␣
↪if missing else " "

        md_print(f"(n_shot: {n_shot}) Found **{len(run)}** checkpoints of␣
↪**{max_num_epochs}** expected. {text_missing}")
```

Found **3** runs with inference outputs out of **3** total filtered runs

Found a total of **6** checkpoints.

(n_shot: 0) Found **15** checkpoints of **15** expected.

(n_shot: 4) Found **15** checkpoints of **15** expected.

(n_shot: 0) Found **15** checkpoints of **15** expected.

(n_shot: 4) Found **15** checkpoints of **15** expected.

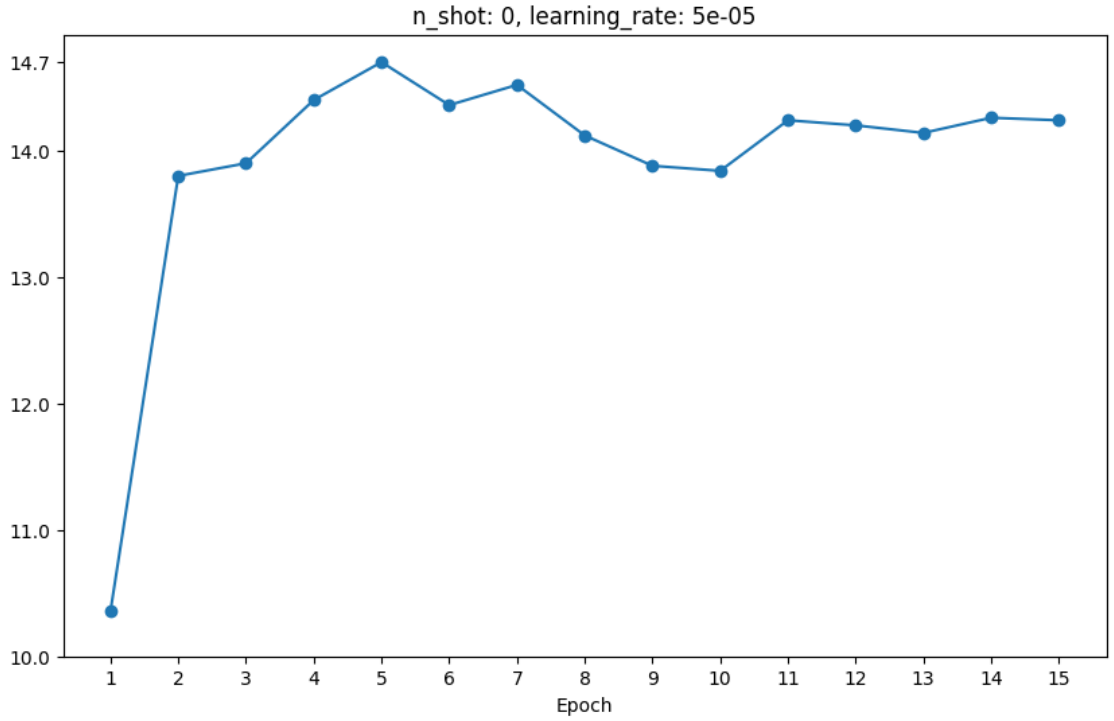
(n_shot: 0) Found **15** checkpoints of **15** expected.

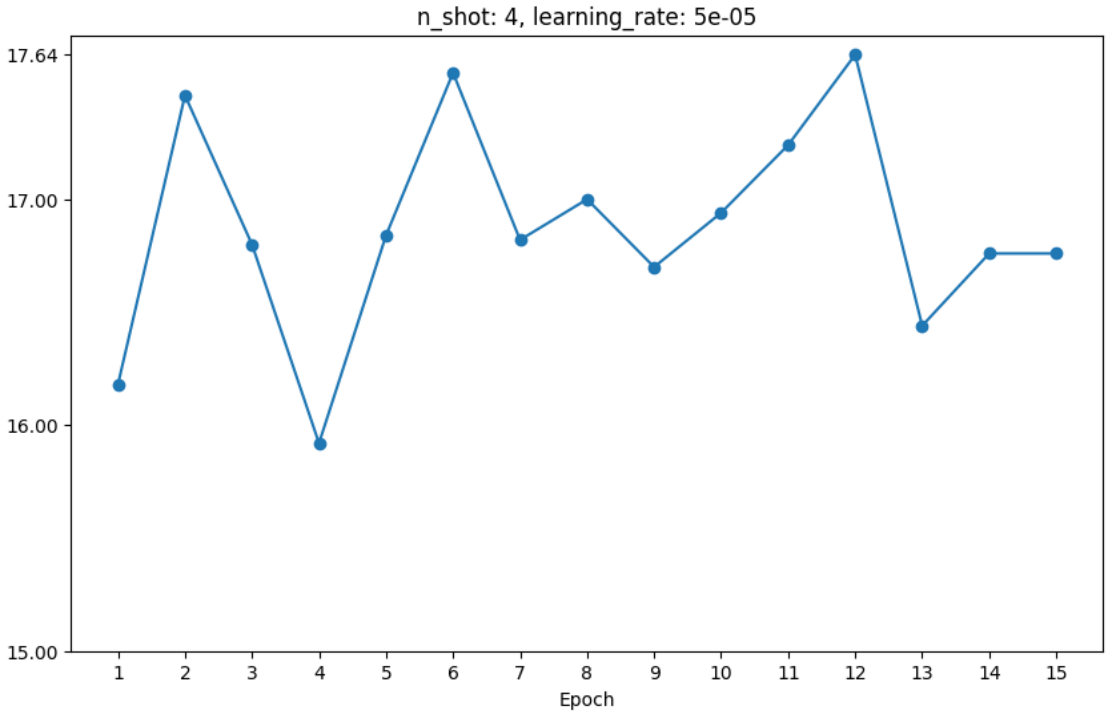
(n_shot: 4) Found **15** checkpoints of **15** expected.

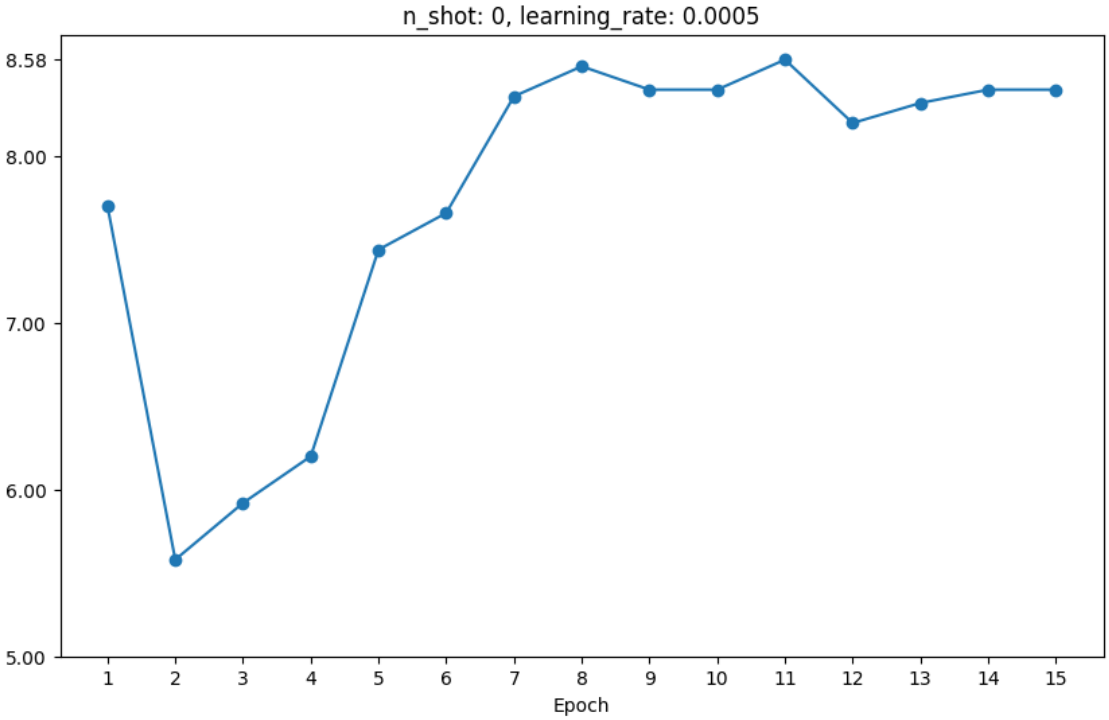
4 Plot the accuracy for the epochs that we do have, per run.

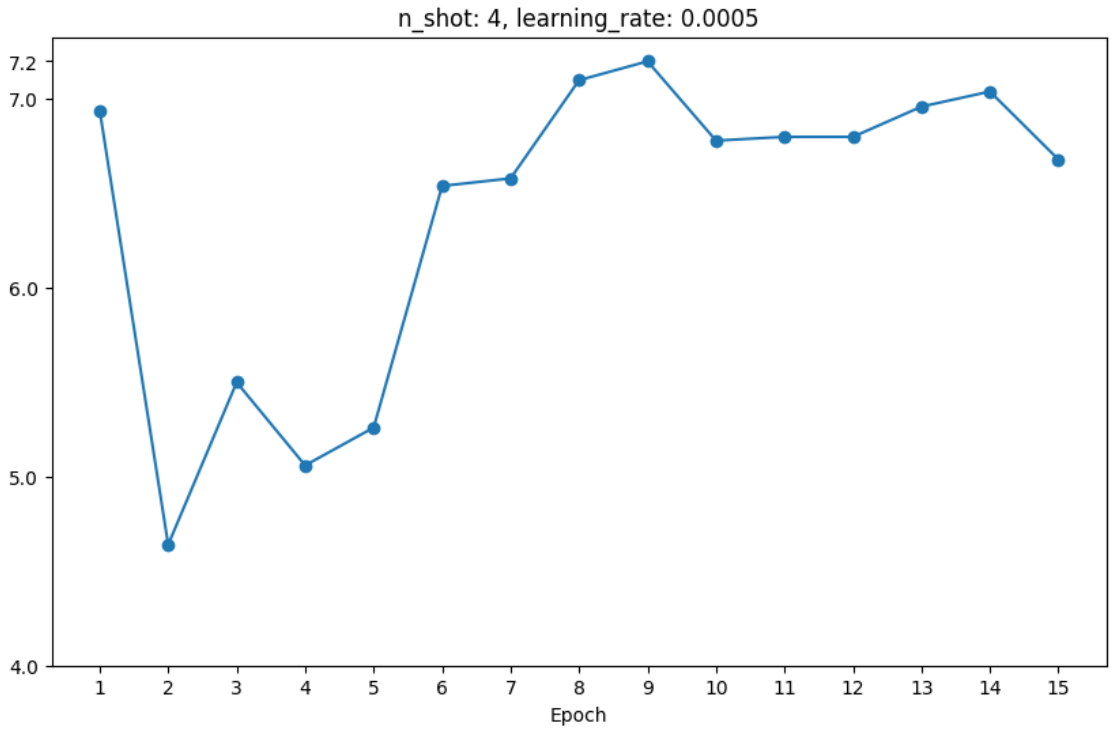
[7]:
```
SHOW_PLOTS = True
if SHOW_PLOTS:
    for n_shot_to_run in filtered_runs_with_outputs:
        for n_shot, run in n_shot_to_run.items():
            by_epoch = {}
            learning_rates = set()
            for ckpt in run:
                results_path = ckpt.results_path
                epoch = ckpt.meta_config["epoch"]
                # Extract results from this
                results_jsons = mit.one(results_path.glob("**/results*.json"))
                with open(results_jsons, "r") as f:
                    parsed = json.load(f)
                by_epoch[epoch] = parsed["results"]["all"]["qem"]
                learning_rate = ckpt.meta_config["cfg"]["learning_rate"]
                learning_rates.add(learning_rate)

            # Create a figure for each run
            plt.figure(figsize=(10, 6))
            plt.title(f"n_shot: {n_shot}, learning_rate: {mit.
↪one(learning_rates)}")

            by_epoch = dict(sorted(by_epoch.items()))
            y = np.array(list(by_epoch.values())) * 100
```

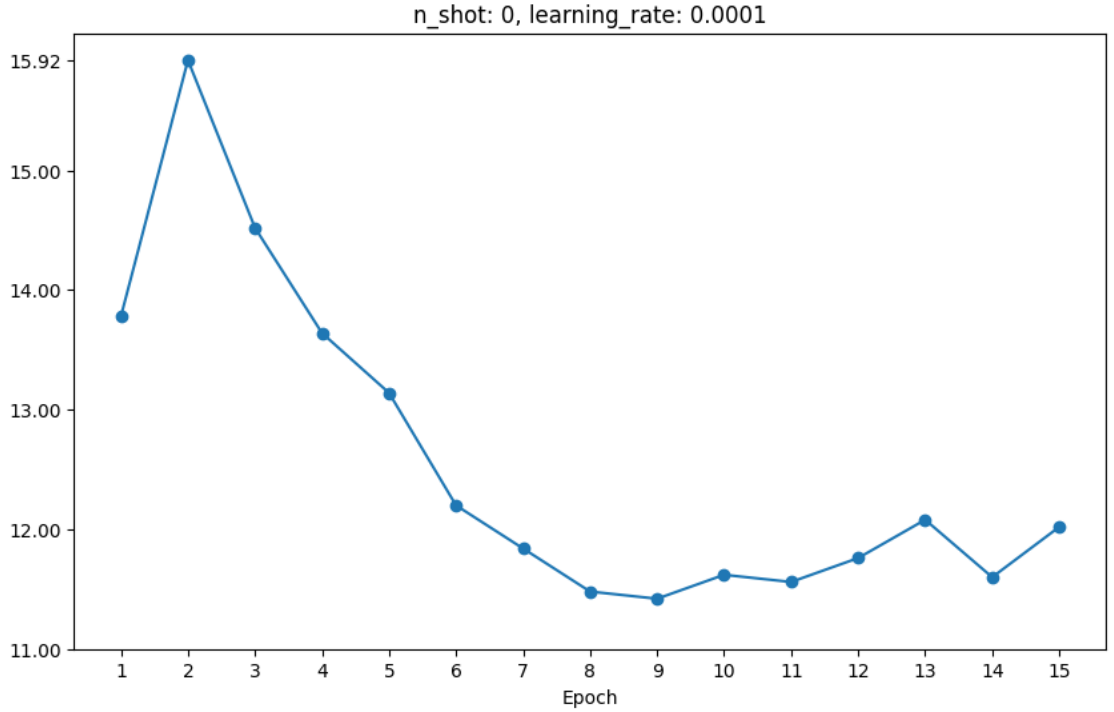
```
            x = np.array(list(by_epoch.keys())) + 1

            # Plot the accuracy for each epoch
            plt.plot(x, y, marker='o')
            plt.xticks(np.arange(1, x.max() + 1))
            plt.yticks(np.arange(int(y.min()), int(y.max()) + 1).tolist() + [y.
↪max()])

            # Add labels and title
            plt.xlabel("Epoch")
            plt.show()
```

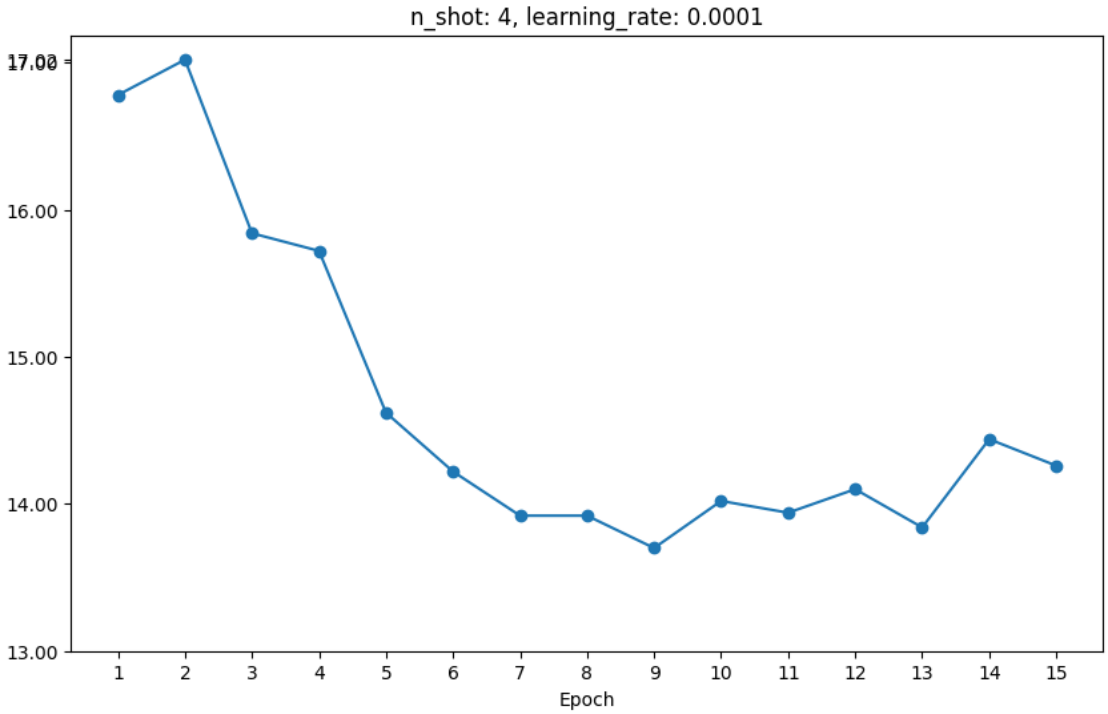



n_shot: 4, learning_rate: 5e-05
17.64
17.00
16.00
15.00
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Epoch


n_shot: 0, learning_rate: 0.0005
8.58
8.00
7.00
6.00
5.00
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Epoch


n_shot: 4, learning_rate: 0.0005
7.2
7.0
6.0
5.0
4.0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Epoch


n_shot: 0, learning_rate: 0.0001
15.92
15.00
14.00
13.00
12.00
11.00
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Epoch



5 We re-compute the the scores with our custom parser.

[8]:
```
import verify
import importlib
importlib.reload(verify)

N_QTY = 100
MAX_NUM_EPOCHS = 15

time_spent = verify.FractionTimeSpent()

output_data = collections.defaultdict(lambda: collections.defaultdict(list))
for i, n_shot_to_runs in enumerate(filtered_runs_with_outputs):
    for n_shot, runs in n_shot_to_runs.items():
        if n_shot == 0:
            for run in runs:
                if run["cfg"]["max_num_epochs"] == MAX_NUM_EPOCHS:
                    parquet_path = mit.one(run.inference_outputs_path.glob("**/
 ↪*.parquet"))
                    results = verify.compute_score(parquet_path, mode=verify.
 ↪Mode.math, time_spent=time_spent, compute_score=True, subset_qty=N_QTY)
                    print(results[0].row(0, named=True))
                    output_data[i][n_shot].append(results)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[8], line 15
     13 if n_shot == 0:
     14     for run in runs:
---> 15         if run[     ]["max_num_epochs"] == MAX_NUM_EPOCHS:
     16             parquet_path = mit.one(run.inference_outputs_path.glob("**/.
 ↪parquet"))
     17             results = verify.compute_score(parquet_path, mode=verify.
 ↪Mode.math, time_spent=time_spent, compute_score=True, subset_qty=N_QTY)

TypeError: 'MetaInfo' object is not subscriptable
```

[ ]:
```

```