willibaldlenthin via Pinboard

Reading List - 2025-12-30

Dec 30, 2025

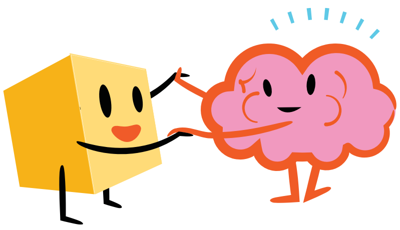
Table of Contents

Source:
https://www.reddit.com/r/LocalLLaMA/comments/1khjrtj/building_llm_workflows_some_observations/

You've been blocked by network security.
To continue, log in to your Reddit account or use your developer token
If you think you've been blocked by mistake, file a ticket below and we'll look into it.
Log in → https://www.reddit.com/login/File a ticket → https://support.reddithelp.com/hc/en-us/requests/new?ticket_form_id=21879292693140

K-8 Math Program | Built by Teachers, for Teaching | Zearn Math

Source: https://about.zearn.org/



Zearn Math logo

Discover Zearn

Multiply the ‘aha’ moments in your classroom.

Zearn is the top-rated math learning platform that helps kids explore concepts, discover meaning, and make sense of math. Free for teachers, always.

Rated

“Strong” (Tier 1) by Evidence for ESSA for proven impact

✓ **1,000+ research-backed K–8 digital lessons**

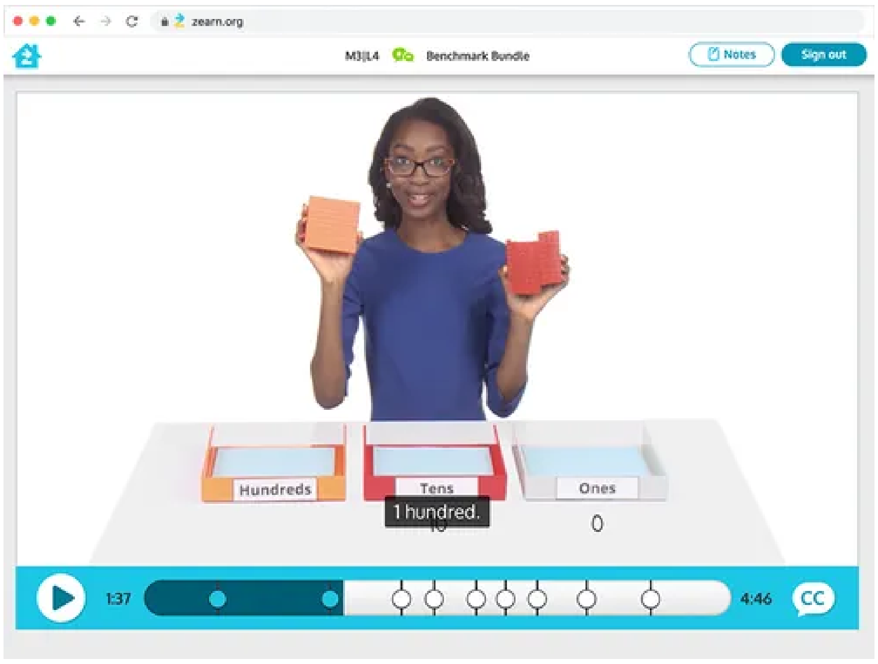
★ **Paper-and-pencil problem solving in every lesson**

Built on the concrete-to pictorial-to-abstract approach

How Zearn works.

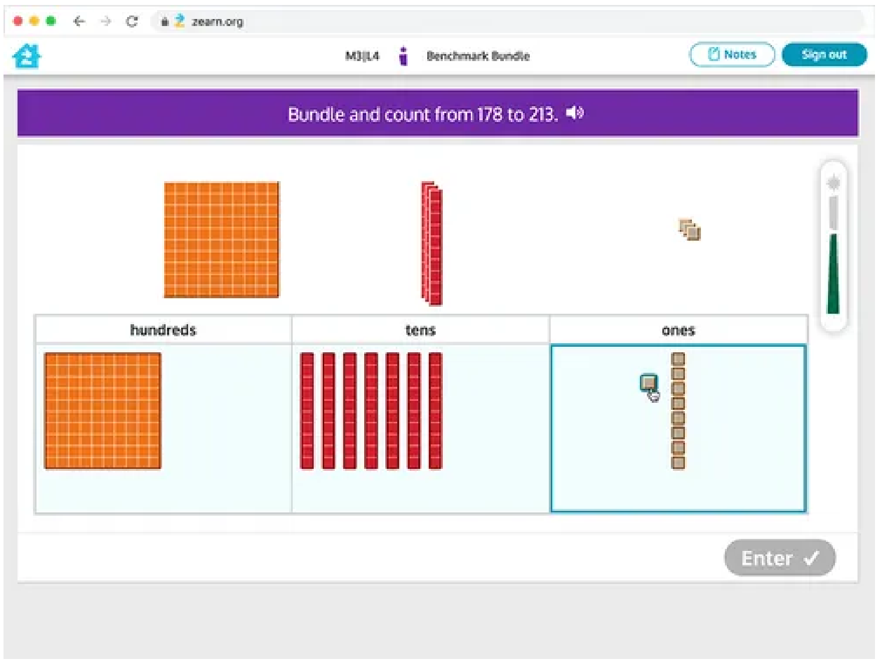
Zearn Math gives kids more opportunity to learn and practice math, always connected to core instruction.

Kids learn math concepts from their teacher, in the classroom.



Instructional videos led by on-screen teachers with visual models

They log in to Zearn and explore grade-level math with onscreen teachers, visual models and digital manipulatives.



Online learning program with built-in differentiated support to help kids learn from their mistakes

When kids struggle, they get built-in differentiated support to help them learn from their mistakes and keep moving forward.

Made

for teaching.

In a fall 2023 survey of over 1,000 teachers, 90% of teachers said they would recommend that other teachers use Zearn Math because of Zearn's connection to core math instruction and built-in support.

Quotes from teachers about how students are engaged with Zearn Math

Quotes from educators who use Zearn Math for online learning

Zearn has helped my students build their math confidence and be able to **engage more during math class.**

6TH GRADE TEACHER
OMAHA, NE

Zearn helps me **differentiate instruction for each student.**

5TH GRADE TEACHER
WILMINGTON, NC

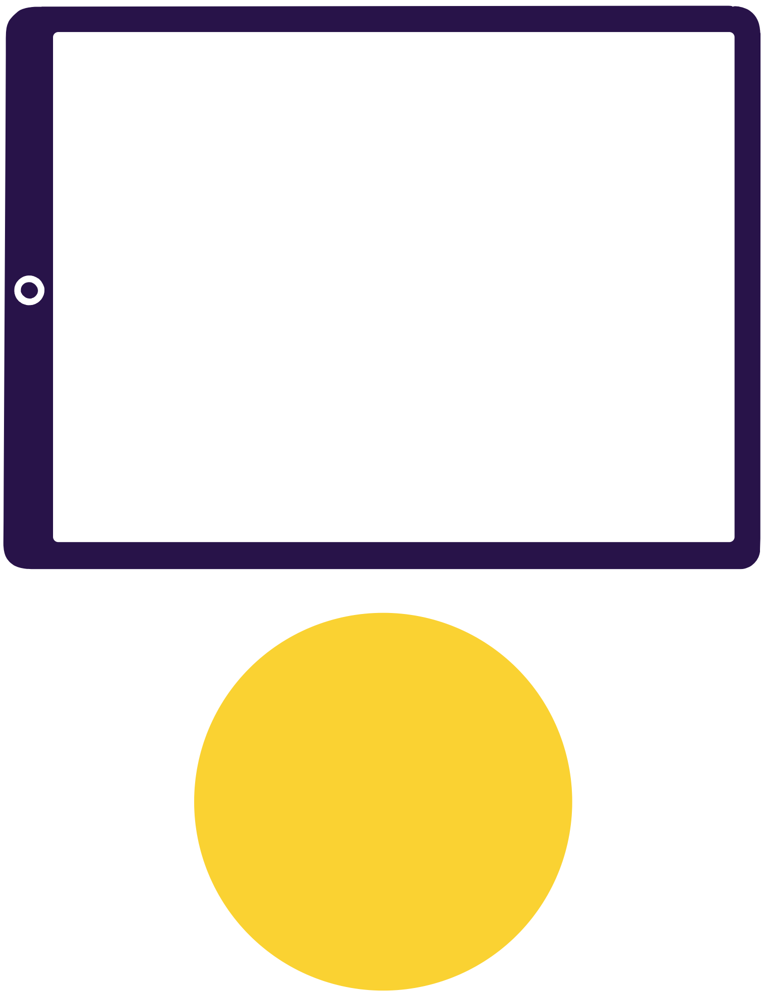
The growth I've seen in my own classroom speaks for itself.

5TH GRADE TEACHER
TEMPLETON, CA

Zearn has tons of resources, **the kids enjoy it, and most of all, they learn.**

4TH GRADE TEACHER
SHREVEPORT, LA

Learning is priceless. It's also free.


Zearn Math digital lessons

For individual teachers and classrooms. *Always.*We’re a nonprofit, so we’re not driven by dollars—everything we do, offer, and make has one goal: to help kids love learning math.

Let's do this.

First, check out the types of accounts we offer, and then let's get started together.

Overengineering PR create with jj – David Crespo

Source: https://crespo.business/posts/overeng-pr-create-jj/

This note is about a script → https://github.com/david-crespo/dotfiles/blob/650549d52c4550031a9d06dac0c7d1e81084de95/bin/jprc.ts I wrote that makes creating a PR marginally more convenient, and how it had to change when I switched to Jujutsu → https://jj-vcs.github.io/jj/latest. It's not an intro to Jujutsu — see the learning resources for that.

Make a PR in six steps, or one

You've written some code and you're ready to make a PR. Getting to the PR create form on GitHub takes a lot of steps:

1. Push your branch
2. Find the repo on GitHub
3. Go to pull requests
4. Click "New pull request"
5. Find the right branch (what did you call it?)
6. Click "Create pull request"

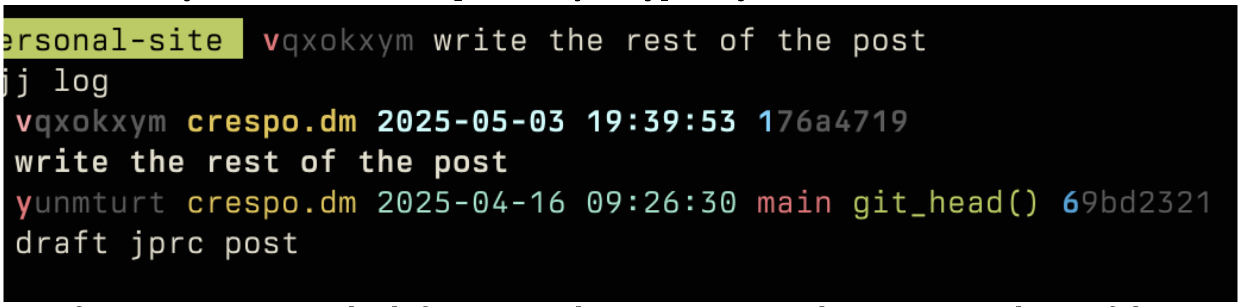
Thanks to the wonderful GitHub CLI → https://cli.github.com/, you can do all that from the terminal in one line:

```
alias gprc='git push -u && gh pr create --web'
```

For git users, this post is over. Use that. I used it for 3 years. It's the best.

In Jujutsu, you create the branch last

If you're using Jujutsu, you can do more or less the same thing, except at this point you don't have a named branch yet. In Jujutsu, you don't start with a branch and work "on" that branch — there is no concept of a "current" branch. You just make commits on top of `main`, and only when it's time to push do you typically create a bookmark.[1]



```
ersonal-site  vqxokxym write the rest of the post
jj log
 vqxokxym crespo.dm 2025-05-03 19:39:53 176a4719
 write the rest of the post
 yunmturt crespo.dm 2025-04-16 09:26:30 main git_head() 69bd2321
 draft jprc post
```

If you've just switched from git, this is annoying because our beautiful `gprc` isn't enough anymore: you have to create a branch too, and naming a branch is slightly more than zero effort, i.e., too much effort.

Many jj users avoid naming the branch by passing `--change <REVSETS>` to `jj git push` → https://jj-vcs.github.io/jj/v0.28.2/cli-reference/#jj-git-push, which tells jj to use the change ID → https://jj-vcs.github.io/jj/v0.28.2/glossary/#change-id, resulting in a branch like `push-urzrzuzsurwx`.[2] Indeed, branch names like that are one of the few signs visible on GitHub that someone is using jj.

Branch names matter, but not that much

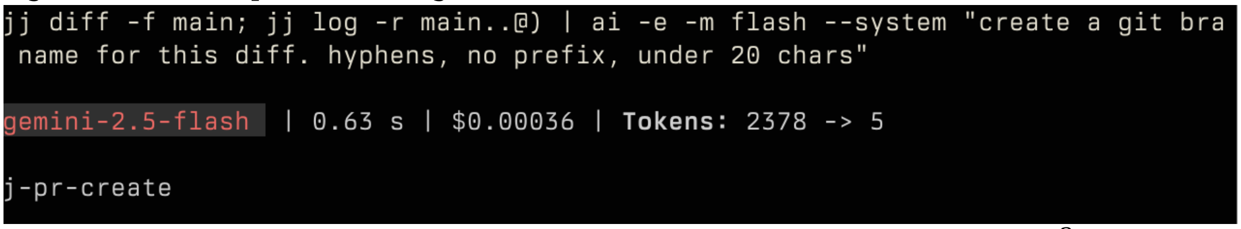
The thing is, meaningful branch names can be useful. When your colleague wants to check out your work locally so they can test it, they're probably using your branch name to find it. You might be using branch names yourself to switch between ongoing streams of work.

Branch names do not have to be perfect to be useful in this way. Unlike PR titles and descriptions (which tell people what you did and why, and which become commit messages on merge), branch names just need to vaguely point to the right change. Naming them is not some great art.

Let's make the computer do it.

Generating a branch name

How can you automatically generate a decent branch name? It sounds impossible. Oh, right: in 2025, impossible things cost three hundredths of a cent.



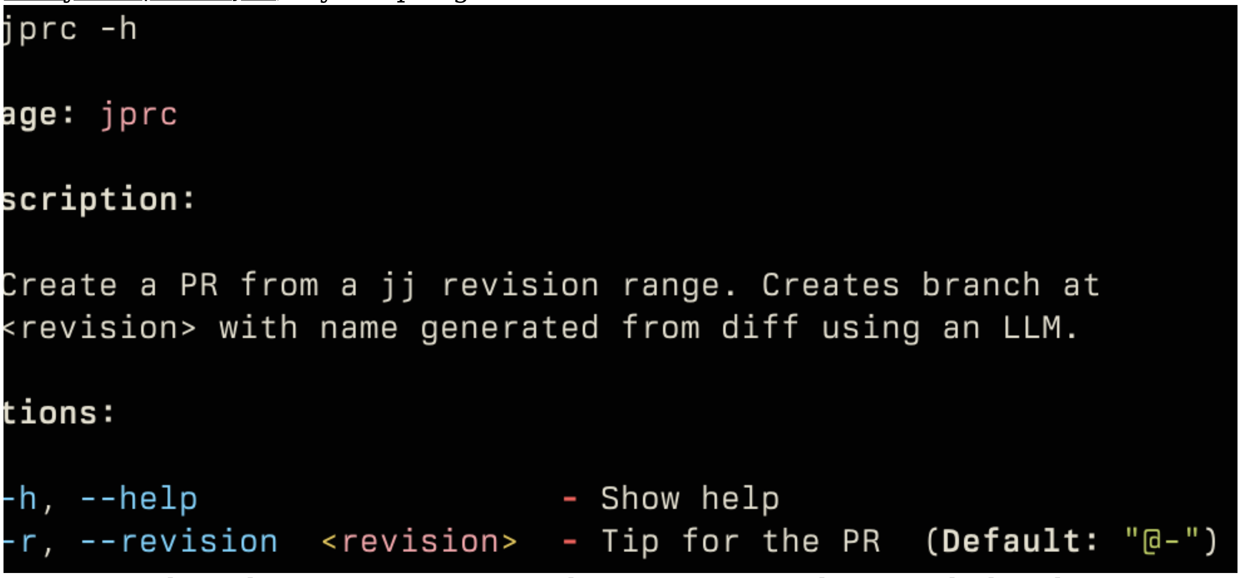
```
jj diff -f main; jj log -r main..@) | ai -e -m flash --system "create a git bra
 name for this diff. hyphens, no prefix, under 20 chars"

gemini-2.5-flash  | 0.63 s | $0.00036 | Tokens: 2378 -> 5

j-pr-create
```

Well, that was easy. I piped the diff and commit log to an LLM CLI[3] that sent it to Gemini 2.5 Flash (the best fast and cheap model at time of writing) and got out a branch name in half a second for $0.00036. The only real trick here is using the LLM CLI as a super simple way of hooking into an LLM and knowing which model to use.[4]

Now let's see what it takes to plug this operation into a usable script.

The final product: `jprc`

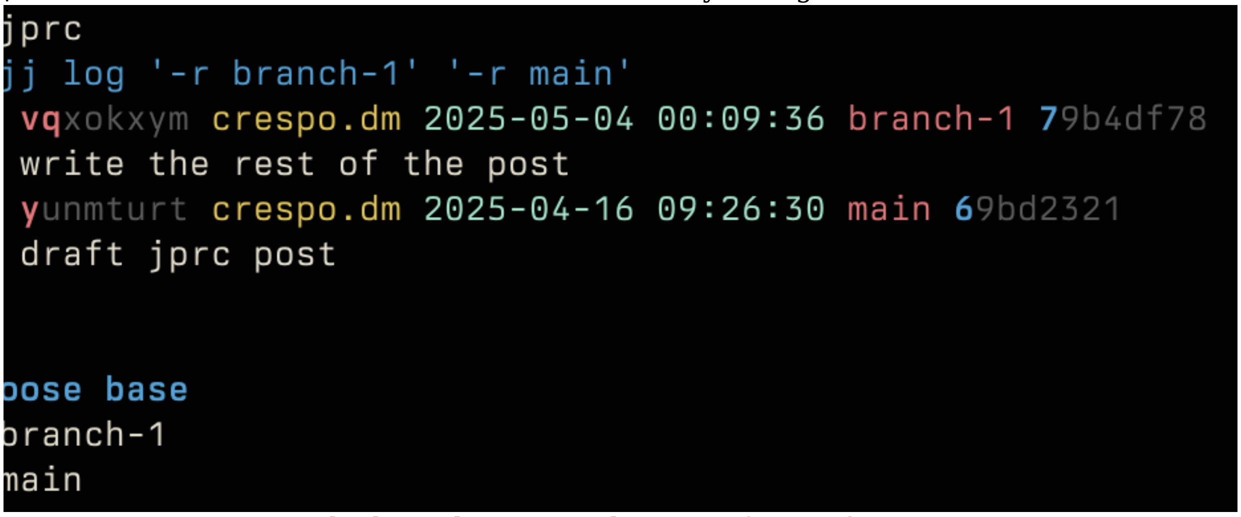
You can see the full `jprc.ts` → https://github.com/david-crespo/dotfiles/blob/650549d5/bin/jprc.ts in my dotfiles, powered by Deno → https://docs.deno.com/runtime/, dax → https://github.com/dsherret/dax, and Cliffy → https://cliffy.io/, my scripting tools of choice.



```
jprc -h

age: jprc

scription:

Create a PR from a jj revision range. Creates branch at
<revision> with name generated from diff using an LLM.

tions:

-h, --help                   - Show help
-r, --revision  <revision>   - Tip for the PR  (Default: "@-")
```

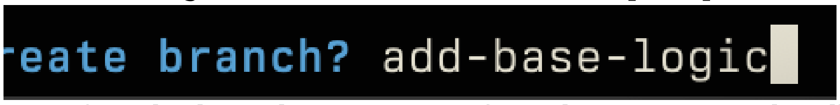
It's 54 lines because it turns out there's more you have to do besides generate the branch name. Half the lines are devoted to helping the user pick a base branch if there are any branches between `main` and the target revision. Sometimes we are making a small

PR on top of a big PR, and if we use the full diff since `main` to generate the branch name, it will be not be specific enough to the small change. We will also pass the base branch to `gh pr create` with `--base` so we don't have to manually change it from `main` in the form.



```
jprc
jj log '-r branch-1' '-r main'
 vqxokxym crespo.dm 2025-05-04 00:09:36 branch-1 79b4df78
 write the rest of the post
 yunmturt crespo.dm 2025-04-16 09:26:30 main 69bd2321
 draft jprc post


oose base
branch-1
main
```

Now we generate the branch name and prompt for confirmation:



```
eate branch? add-base-logic
```

After the branch name is confirmed, we create it locally, push it, and run `gh pr create` as before.

Here's the core logic:

```
// 1. Pick a base branch
const base = await pickBase(r)

// 2. Make sure base is on the remote
const result = await $`jj bookmark list --remote origin ${base}`.text()
if (!result) throw new ValidationError(`Base '${base}' not found on origin.`)

console.log(`\nCreating PR with base %c${base}\n`, "color: #ff6565")

// 3. Print the commit log
const range = `${base}..${r}`
await $`jj log -r ${range}`.printCommand()

// 4. Generate a branch name and confirm it (allowing editing)
const generated = await $`jj diff -r ${range}; jj log -r ${range}`
  .pipe($`ai --system "${prompt}" -m flash --raw --ephemeral`)
  .text()

// 5. Confirm branch name, allowing editing
const opts = { noClear: true, default: generated.trim() }
const bookmark = await $.prompt("\nCreate branch?", opts)

// 6. Push the branch
await $`jj git push --named ${bookmark}=${r}`.printCommand()

// 7. Create the PR
await $`gh pr create --head ${bookmark} --base ${base} --web`.printCommand()
```

Isn’t the LLM overkill?

It’s natural to wonder whether bringing LLMs into this little moment is overkill. But I think that reaction is based partly on an intuition I’ve shown to be wrong: that calling an LLM is complicated, costly, or risky. With the right tools and in the right scenario, LLMs fit so cleanly into Unix pipelines (text in, text out) that building a silly idea like this into a personal tool is trivial. That triviality makes it equally easy to ask “why bother?” and “why not?” It makes it like asking: isn’t bringing `awk` into this script overkill? Maybe, but only if it’s doing absolutely nothing for you.

It’s worth thinking through why the downside risk of a potentially unreliable helper is so small in this particular case. The stakes are low, the output is human-reviewed (or otherwise easy to verify[5]), and it’s easy to recover if the branch name is bad: you just make up a different name. If any of those things weren’t true, the downside risk might be intolerable. It also needs to be fast and cheap, and to be good enough that you don’t have to recover very often — if the branch name was bad half the time, it wouldn’t be worth it. All of these are contingent factors that need to be evaluated case by case.

The LLM integration is one line of code, the one that starts with `const generated = ...`. If we take that line out, instead prompting the user for the branch name, the script would still be worth using for the other things it does. In fact, that is how an earlier version → https://github.com/david-crespo/dotfiles/blob/29b3a9cfba5c8792403f4ffbd5f889ce4cec99c5/bin/jprc.ts of `jprc` worked. With such a simple integration, it’s easy to try generating the branch name, evaluate whether it’s useful, and take it out if not.

Can you generate the PR title and body? Should you?

You could generate a PR title and body too and pass them to `gh pr create` with `--title` and `--body`. I wouldn’t — my PR descriptions are a lot more about the why than the what, and that’s rarely represented in the diff or the commit messages. I would spend more time reading and rewriting whatever it produced than I would save from having it prefilled — this violates the requirement mentioned above that the output be easy to verify.

If you included the text of an issue that is closed by the PR, maybe the model could explain how the change closes the issue, but even that sounds dicey. Coding agents like Cursor or Claude Code probably have a better shot at generating decent PR bodies based on the conversation that produced the diff.

Jujutsu resources

- The jj README → https://github.com/jj-vcs/jj?tab=readme-ov-file#introduction — I actually like this better than the intro on their site
- Steve’s Jujutsu Tutorial → https://steveklabnik.github.io/jujutsu-tutorial/ — everybody loves Steve
- Git and jujutsu: in miniature → https://lottia.net/notes/0013-git-jujutsu-miniature.html — the post that made it finally click for me
- What I’ve learned from jj → https://zerowidth.com/2025/what-ive-learned-from-jj/ — the “what’s cool about jj” post I wish I’d written

Footnotes

1. Branches are called bookmarks in Jujutsu. The name comes from Mercurial → https://wiki.mercurial-scm.org/Bookmarks. The jj devs changed the name from branches to bookmarks while I was learning it, and for whatever reason the new name really helped me get past my git-based expectations. ↩
2. You can configure → https://jj-vcs.github.io/jj/v0.28.2/config/#prefix-for-generated-bookmarks-on-push the prefix. People like to add their username to the front, like `david-crespo/push-`. ↩

3. I’m using my own LLM CLI → https://github.com/david-crespo/llm-cli here, but it’s pretty stripped down and tailored to my needs, so I usually recommend Simon Willison’s `llm` → https://llm.datasette.io/. ↩
4. I tried other light models like Claude 3.5 Haiku, GPT-4.1 mini, Llama 3.3 70B (through both Groq and Cerebras), and Llama 4 Maverick and Scout. They were mostly fine, though none was as consistently good as Gemini 2.5 Flash. Llama 4 Scout is clearly not good enough: at one point it generated the branch name `branch-names-are-hard`. ↩
5. I have in mind something like how typechecking and tests are a constraint on how wrong LLM-generated code can be. ↩

Mintlify - The Intelligent Documentation Platform

Source: https://mintlify.com/

Built for the intelligence age

Integrate AI into every part of your docs lifecycle. Woven into how your knowledge is written, maintained, and understood by both users and LLMs.

Qodo/Qodo-Embed-1-7B · Hugging Face

Source: https://huggingface.co/Qodo/Qodo-Embed-1-7B

Qodo-Embed-1 is a state-of-the-art code embedding model designed for retrieval tasks in the software development domain. It is offered in two sizes: lite (1.5B) and medium (7B). The model is optimized for natural language-to-code and code-to-code retrieval, making it highly effective for applications such as code search, retrieval-augmented generation (RAG), and contextual understanding of programming languages. This model outperforms all previous open-source models in the COIR and MTEB leaderboards, achieving best-in-class performance with a significantly smaller size compared to competing models.

Languages Supported:

- Python
- C++
- C#
- Go
- Java
- Javascript
- PHP
- Ruby
- Typescript

Model Information

- Model Size: 7B
- Embedding Dimension: 3584
- Max Input Tokens: 32k

Requirements

```
transformers>=4.39.2
flash_attn>=2.5.6
```

Usage

Sentence Transformers

```
from sentence_transformers import SentenceTransformer

# Download from the 🤗 Hub
model = SentenceTransformer("Qodo/Qodo-Embed-1-7B")
# Run inference
sentences = [
    'accumulator = sum(item.value for item in collection)',
    'result = reduce(lambda acc, curr: acc + curr.amount, data, 0)',
    'matrix = [[i*j for j in range(n)] for i in range(n)]'
]
embeddings = model.encode(sentences)
print(embeddings.shape)
# [3, 1536]

# Get the similarity scores for the embeddings
similarities = model.similarity(embeddings, embeddings)
```

```
print(similarities.shape)
# [3, 3]
```

Transformers

```
import torch
import torch.nn.functional as F

from torch import Tensor
from transformers import AutoTokenizer, AutoModel


def last_token_pool(last_hidden_states: Tensor,
                 attention_mask: Tensor) -> Tensor:
    left_padding = (attention_mask[:, -1].sum() == attention_mask.shape[0])
    if left_padding:
        return last_hidden_states[:, -1]
    else:
        sequence_lengths = attention_mask.sum(dim=1) - 1
        batch_size = last_hidden_states.shape[0]
        return last_hidden_states[torch.arange(batch_size, device=last_hid-
den_states.device), sequence_lengths]


# Each query must come with a one-sentence instruction that describes the task
queries = [
      'how to handle memory efficient data streaming',
      'implement binary tree traversal'
  ]

documents = [
        """def process_in_chunks():
            buffer = deque(maxlen=1000)
            for record in source_iterator:
                buffer.append(transform(record))
                if len(buffer) >= 1000:
                    yield from buffer
                    buffer.clear()""",

        """class LazyLoader:
            def __init__(self, source):
                self.generator = iter(source)
                self._cache = []

            def next_batch(self, size=100):
                while len(self._cache) < size:
                    try:
                        self._cache.append(next(self.generator))
                    except StopIteration:
                        break
                return self._cache.pop(0) if self._cache else None""",

        """def dfs_recursive(root):
            if not root:
                return []
            stack = []
            stack.extend(dfs_recursive(root.right))
            stack.append(root.val)
            stack.extend(dfs_recursive(root.left))
            return stack"""
    ]
```

```
input_texts = queries + documents

tokenizer = AutoTokenizer.from_pretrained('Qodo/Qodo-Embed-1-7B',
trust_remote_code=True)
model = AutoModel.from_pretrained('Qodo/Qodo-Embed-1-7B', trust_remote_code=True)

max_length = 8192

# Tokenize the input texts
batch_dict = tokenizer(input_texts, max_length=max_length, padding=True, trunca-
tion=True, return_tensors='pt')
outputs = model(**batch_dict)
embeddings = last_token_pool(outputs.last_hidden_state, batch_dict['atten-
tion_mask'])

# normalize embeddings
embeddings = F.normalize(embeddings, p=2, dim=1)
scores = (embeddings[:2] @ embeddings[2:].T) * 100
print(scores.tolist())
```

License

marimo | a next-generation Python notebook

By marimo
Source: https://marimo.io/

The future of Python notebooks is here

Transform data, train models, and run SQL queries with marimo — feels like an AI-native reactive notebook, stored as Git-friendly reproducible Python. Seamlessly run as scripts and apps. All open source.

`$ $`

The notebook, reinvented

marimo feels like a notebook but is stored as pure Python program that's Git-friendly, reusable as a module, executable as a script, shareable as an app, reproducible in execution and packaging, and designed for data (with SQL and LLM support built-in).

And it's all open source.

Made with marimo

Make just about anything. Here are example notebooks to jog your imagination.

Trusted by teams worldwide

Learn why marimo is transformational to organizations large and small.

Join the community

Find out what others are building
Get notified about new features and updates

Mixture-of-Agents Enhances Large Language Model Capabilities

Source: https://arxiv.org/html/2406.04692v1

Junlin Wang
Duke University
Together AI
junlin.wang2@duke.edu
&Jue Wang
Together AI
jue@together.ai
&Ben Athiwaratkun
Together AI
ben@together.ai
&Ce Zhang
University of Chicago
Together AI
cez@uchicago.edu
&James Zou
Stanford University
Together AI
jamesz@stanford.edu

Abstract

Recent advances in large language models (LLMs) demonstrate substantial capabilities in natural language understanding and generation tasks. With the growing number of LLMs, how to harness the collective expertise of multiple LLMs is an exciting open direction. Toward this goal, we propose a new approach that leverages the collective strengths of multiple LLMs through a Mixture-of-Agents (MoA) methodology. In our approach, we construct a layered MoA architecture wherein each layer comprises multiple LLM agents. Each agent takes all the outputs from agents in the previous layer as auxiliary information in generating its response. MoA models achieves state-of-art performance on AlpacaEval 2.0, MT-Bench and FLASK, surpassing GPT-4 Omni. For example, our MoA using only open-source LLMs is the leader of AlpacaEval 2.0 by a substantial gap, achieving a score of 65.1% compared to 57.5% by GPT-4 Omni.[11]1Our code can be found in: https://github.com/togethercomputer/moa.

1 Introduction

Large language models (LLMs) *(Zhang et al., 2022a → https://arxiv.org/html/2406.04692v1#bib.bib42; Chowdhery et al., 2022 → https://arxiv.org/html/2406.04692v1#bib.bib6; Touvron et al., 2023a → https://arxiv.org/html/2406.04692v1#bib.bib29; Team et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib27; Brown et al., 2020 → https://arxiv.org/html/2406.04692v1#bib.bib2; OpenAI, 2023 → https://arxiv.org/html/2406.04692v1#bib.bib20)* have significantly advanced the field of natural language understanding and generation in recent years. These models are pretrained on vast amounts of data and subsequently aligned with human preferences to generate helpful and coherent outputs *(Ouyang et al., 2022 → https://arxiv.org/html/2406.04692v1#bib.bib21)*. However, despite the plethora of LLMs and their impressive achievements, they still face inherent constraints on model size and training data. Further scaling up these models is exceptionally costly, often requiring extensive retraining on several trillion tokens.

At the same time, different LLMs possess unique strengths and specialize in various tasks aspects. For instance, some models excel at complex instruction following *(Xu et al., 2023a → https://arxiv.org/html/2406.04692v1#bib.bib36)* while others may be better suited for code generation *(Roziere et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib24; Guo et al., 2024 → https://arxiv.org/html/2406.04692v1#bib.bib11)*. This diversity in skill sets among different LLMs presents an intriguing question: Can we harness the collective expertise of multiple LLMs to create a more capable and robust model?

Our answer to this question is Yes. We identify an inherent phenomenon we term the collaborativeness of LLMs — wherein an LLM tends to generate better responses when presented with outputs from other models, even if these other models are less capable by itself. Figure˜1 → https://arxiv.org/html/2406.04692v1#S1.F1 showcases the LC win rate on the AlpacaEval 2.0 benchmark *(Dubois et al., 2024 → https://arxiv.org/html/2406.04692v1#bib.bib9)* for 6 popular LLMs.

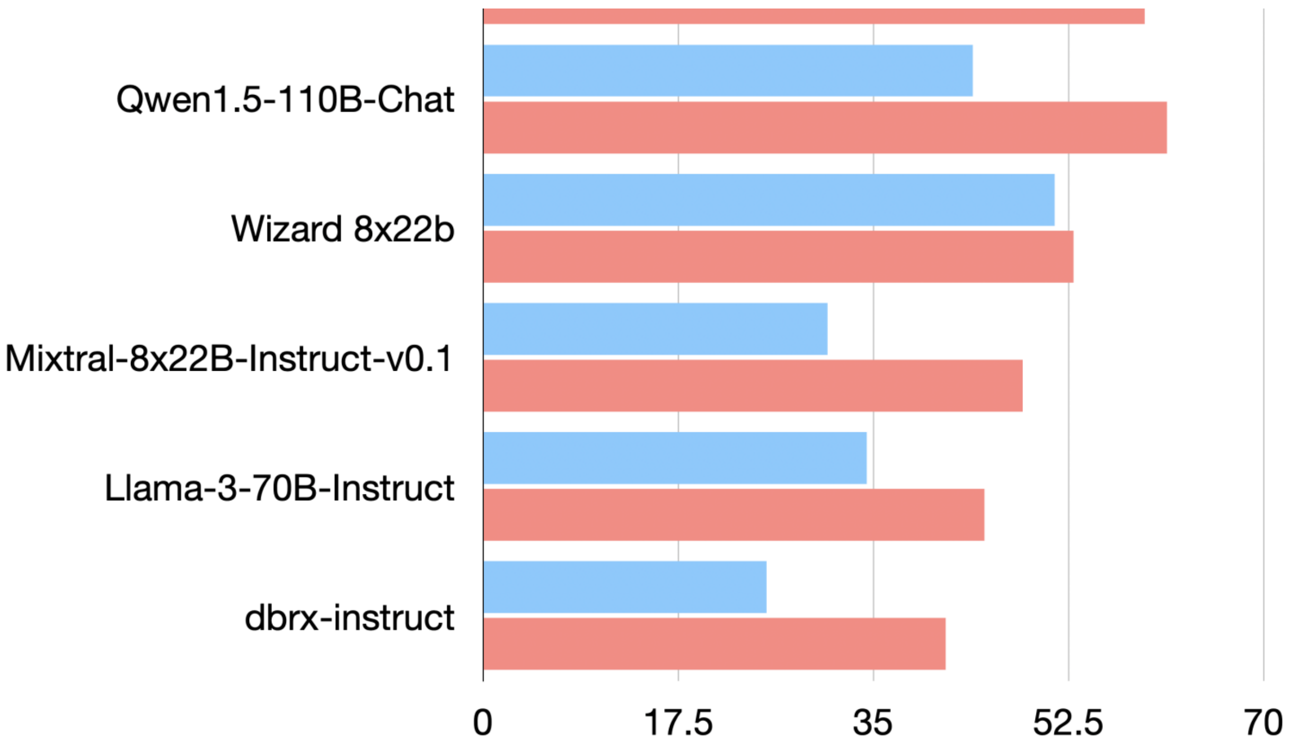


Figure 1: AlpacaEval 2.0 LC win rates improve when provided with responses from other models.

When these models are provided with answers generated independently by these models, their LC win rates significantly improve. This indicates that the collaborativeness phenomenon is widespread among LLMs. Remarkably, this improvement occurs even when the auxiliary responses provided by the other models are of lower quality than what an individual LLM could generate independently.

Based on this finding, this paper introduces a Mixture-of-Agents (MoA) methodology that leverages multiple LLMs to iteratively enhance the generation quality. The structure of MoA is illustrated in Figure˜2 → https://arxiv.org/html/2406.04692v1#S1.F2. Initially, LLMs in the first layer, denoted as agents $A_{1,1}, \ldots A_{1,n}$ independently generate responses to a given prompt. These responses are then presented to agents in the next layer $A_{2,1}, \ldots A_{2,n}$ (which may reuse a model from the first layer) for further refinement. This iterative refinement process continues for several cycles until obtaining a more robust and comprehensive response.

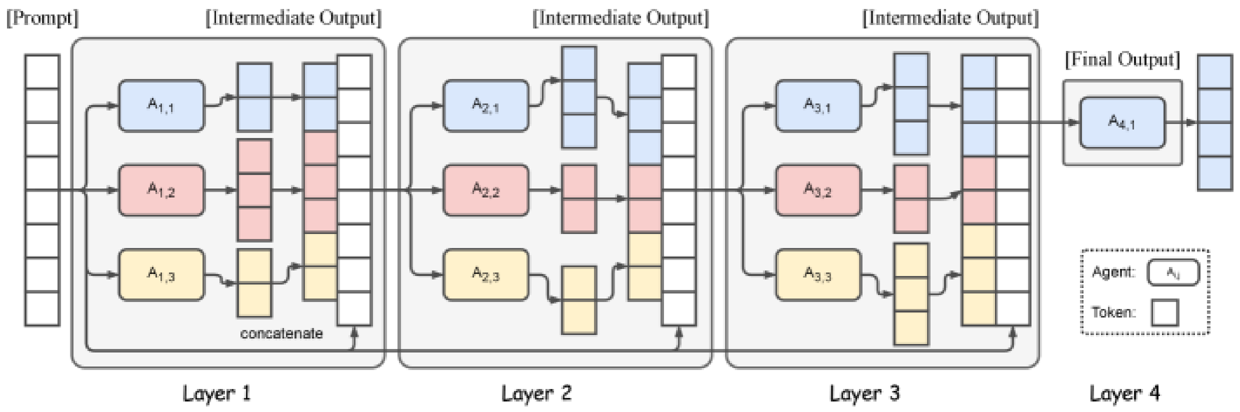


Figure 2: Illustration of the Mixture-of-Agents Structure. This example showcases 4 MoA layers with 3 agents in each layer. The agents here can share the same model.

To ensure effective collaboration among models and improve overall response quality, careful selection of LLMs for each MoA layer is crucial. This selection process is guided by two primary criteria: (a) Performance Metrics: The average win rate of models in layer i plays a significant role in determining their suitability for inclusion in layer $i + 1$. Therefore, selecting models based on their demonstrated performance metrics ensures higher-quality outputs. (b) Diversity Considerations: The diversity of model outputs is also crucial. Responses generated by heterogeneous models contribute significantly more than those produced by the same model as we show later in section˜3.3 → https://arxiv.org/html/2406.04692v1#S3.SS3. By leveraging these criteria — performance and diversity — MoA aims to mitigate individual model deficiencies and enhance overall response quality through collaborative synthesis.

We conduct comprehensive evaluations using AlpacaEval 2.0, MT-Bench *(Zheng et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib44)*, FLASK *(Ye et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib40)* benchmarks for assessing the response quality across various dimensions. The results demonstrate substantial improvements with our proposed method, achieving a new SOTA win rate of 65.8% on AlpacaEval 2.0 compared to the previous best of 57.5% achieved by GPT-4 Omni.

The contributions of this work are summarized as follows: (1) Novel framework: we propose a Mixture-of-Agents framework designed to leverage the strengths of multiple LLMs, thereby improving their reasoning and language generation capabilities. (2) Finding of collaborativeness of language models: we highlight the inherit collaborativeness among LLMs, where models tend to generate better quality responses when they have access to outputs from other models, even if those outputs are of lower quality. (3) State-of-the-art LLM performance: we conducted extensive experiments using multiple highly-competitive benchmarks such as AlpacaEval 2.0, MT-Bench, and FLASK; our MoA framework achieves state-of-the-art performance on these benchmarks.

2 Mixture-of-Agents Methodology

In this section, we present our proposed methodology for leveraging multiple models to achieve boosted performance. We begin by demonstrating that LLMs possess collaborativeness and thus can improve their responses based on the outputs of other models. Following this, we introduce the Mixture-of-Agents methodology and discuss its design implications.

2.1 Collaborativeness of LLMs

We begin by demonstrating the collaborativeness of LLMs, specifically their ability to generate higher quality responses when they can reference outputs from other models. As we have shown in the introduction and Figure˜1 → https://arxiv.org/html/2406.04692v1#S1.F1, many of today's available LLMs exhibit this collaborative capability.

An important pathway to extract maximum benefits from collaboration of multiple LLMs is to characterize how different models are good at in various aspects of collaboration. During the collaboration process, we can categorize LLMs into two distinct roles:

Proposers excel at generating useful reference responses for use by other models. While a good proposer may not necessarily produce responses with high scores by itself, it should offer more context and diverse perspectives, ultimately contributing to better final responses when used by an aggregator.

Aggregators are models proficient in synthesizing responses from other models into a single, high-quality output. An effective aggregator should maintain or enhance output quality even when integrating inputs that are of lesser quality than its own.

Section˜3.3 → https://arxiv.org/html/2406.04692v1#S3.SS3 empirically validate the roles of aggregators and proposers. Specifically, we show that many LLMs possess capabilities both as aggregators and proposers, while certain models displayed specialized proficiencies in distinct roles. GPT-4o, Qwen1.5, LLaMA-3 emerged as a versatile model effective in both assisting and aggregating tasks. In contrast, WizardLM demonstrated excellent performance as an proposer model but struggled to maintain its effectiveness in aggregating responses from other models.

Given that an aggregator can generate higher-quality responses by building upon outputs from other models, we propose further enhancing this collaborative potential by introducing additional aggregators. One intuitive idea is to replicate the exercise with multiple aggregators — initially using several to aggregate better answers and then re-aggregating these aggregated answers. By incorporating more aggregators into the process, we can iteratively synthesize and refine the responses, leveraging the strengths of multiple models to produce superior outcomes. This leads to the design of our proposed Mixture-of-Agents.

2.2 Mixture-of-Agents

The structure of MoA is illustrated in Figure˜2 → https://arxiv.org/html/2406.04692v1#S1.F2. It has l layers and each layer-i consists of n LLMs, denoted by $A_{i,1}$, $A_{i,2}$, ..., $A_{i,n}$. It is important to note that LLMs can be reused either within the same layer or across different layers. When many LLMs in a layer are identical, this configuration leads to a special structure that corresponds to a model generating multiple possibly different outputs (due to the stochasticity of temperature sampling). We refer to this setting as single-proposer, where only a sparse subset of models are activated.
Here, each LLM $A_{i,j}$ processes an input text and generates its continuation. Our method does not require any fine-tuning and only utilizes the interface of prompting and generation of LLMs. Formally, given an input prompt x_1, the output of i-th MoA layer y_i can be expressed as follows:

$$y_i = \oplus_{j=1}^{n} [A_{i,j}(x_i)] + x_1, x_{i+1} = y_i \qquad (1)$$

where $+$ here means concatenation of texts; $\oplus$ means application of the Aggregate-and-Synthesize prompt shown in Table˜1 → https://arxiv.org/html/2406.04692v1#S2.T1 to these model outputs.

Table 1: Aggregate-and-Synthesize Prompt to integrate responses from other models.

You have been provided with a set of responses from various open-source models to the latest user query. Your task is to synthesize these responses into a single, high-quality response. It is crucial to critically evaluate the information provided in these responses, recognizing that some of it may be biased or incorrect. Your response should not simply replicate the given answers but should offer a refined, accurate, and comprehensive reply to the instruction. Ensure your response is well-structured, coherent, and adheres to the highest standards of accuracy and reliability.
Responses from models:
1. [Model Response from $A_{i,1}$]
2. [Model Response from $A_{i,2}$]
...
n. [Model Response from $A_{i,n}$]

In practice, we do not need to concatenate prompt and all model responses so only one LLM is needed to be used in the last layer. Therefore, we use the output of an LLM from the l-th layer ($A_{l,1}(x_l)$) as the final output and evaluate the metrics based on it.

2.3 Analogy to Mixture-of-Experts

Mixture-of-Experts (MoE) *(Shazeer et al., 2017 → https://arxiv.org/html/2406.04692v1#bib.bib25)* is a prominent and well-established technique in machine learning where multiple expert networks specialize in different skill sets. The MoE approach has shown significant success across various applications due to its ability to leverage diverse model capabilities for complex problem-solving tasks. Our MoA method draws inspiration from this methodology. A typical MoE design consists of a stack of layers known as MoE layers. Each layer comprises a set of n expert networks alongside a gating network and includes residual connections for improved gradient flow. Formally, for layer i, this design can be expressed as follows:

$$y_i = \sum_{j=1}^{n} G_{i,j}(x_i) E_{i,j}(x_i) + x_i \tag{2}$$

where $G_{i,j}$ represents the output from the gating network corresponding to expert j, and $E_{i,j}$ denotes the function computed by expert network j. The leverage of multiple experts allows the model to learn different skill sets and focus on various aspects of the task at hand.
From a high-level perspective, our proposed MoA framework extends the MoE concept to the model level by operating at the model level rather than at the activation level. Specifically, our MoA approach leverages LLMs and operates entirely through the prompt interface rather than requiring modifications to internal activations or weights. This means that instead of having specialized sub-networks within a single model like in MoE, we utilize multiple full-fledged LLMs across different layers. Note that in our approach, we consolidate the roles of the gating network and expert networks using a LLM, as the intrinsic capacity of LLMs allows them to effectively regularize inputs by interpreting prompts and generating coherent outputs without needing external mechanisms for coordination.

Moreover, since this method relies solely on prompting capabilities inherent within off-the-shelf models: (1) It eliminates computational overhead associated with fine-tuning; (2) It provides flexibility and scalability: our method can be applied to the latest LLMs regardless of their size or architecture.

3 Evaluation

This section presents a comprehensive evaluation of our proposed MoA. Our findings show that:

1. 1.
   We achieve significant improvements on AlpacaEval 2.0, MT-Bench, and FLASK benchmarks. Notably, with open-source models only, our approach outperforms GPT-4o on AlpacaEval 2.0 and FLASK.
2. 2.
   We conduct extensive experiments to provide better understandings of the internal mechanism of MoA.
3. 3.
   Through a detailed budget analysis, several implementations of MoA can deliver performance comparable to GPT-4 Turbo while being 2 × more cost-effective.

3.1 Setup

Benchmarks

We mainly evaluate models on AlpacaEval 2.0 *(Dubois et al., 2024 → https://arxiv.org/html/2406.04692v1#bib.bib9)*, a leading benchmark for assessing the alignment of LLMs with human preferences. It contains 805 instructions representative of real use cases. Each model's response is directly compared against that of the GPT-4 (gpt-4-1106-preview), with a GPT-4-based evaluator determining the likelihood of preferring the evaluated model's response. To ensure fairness, the evaluation employs length-controlled (LC) win rates, effectively neutralizing length bias.^{22}2This metric tracks closely with human preferences, achieving a Spearman correlation of 0.98 with actual human evaluations *(Dubois et al., 2024 → https://arxiv.org/html/2406.04692v1#bib.bib9)*.

Additionally, we also evaluate on MT-Bench *(Zheng et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib44)* and FLASK *(Ye et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib40)*. MT-Bench uses GPT-4 to grade and give a score to model's answer. FLASK, on the other hand, offers a more granular evaluation with 12 skill-specific scores.

Models

In our study, we constructed our default MoA by using only open-source models to achieve competitive performance. The models included are: Qwen1.5-110B-Chat *(Bai et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib1)*, Qwen1.5-72B-Chat, WizardLM-8x22B *(Xu et al., 2023a → https://arxiv.org/html/2406.04692v1#bib.bib36)*, LLaMA-3-70B-Instruct *(Touvron et al., 2023b → https://arxiv.org/html/2406.04692v1#bib.bib30)*, Mixtral-8x22B-v0.1 *(Jiang et al., 2024 → https://arxiv.org/html/2406.04692v1#bib.bib14)*, dbrx-instruct *(The Mosaic Research Team, 2024 → https://arxiv.org/html/2406.04692v1#bib.bib28)*. We construct 3 MoA layers and use the same set of models in each MoA layer. We use Qwen1.5-110B-Chat as the aggregator in the last layer. We also developed a variant called MoA w/ GPT-4o, which prioritizes high-quality outputs by using GPT-4o as the aggregator in the final MoA layer. Another variant, MoA-Lite, emphasizes cost-effectiveness. It uses the same set of models as proposers but includes only 2 MoA layers and employs Qwen1.5-72B-Chat as the aggregator. This makes it more cost-effective than GPT-4o while achieving a 1.8% improvement in quality on AlpacaEval 2.0. We ensure strict adherence to the licensing terms of all models utilized in this research. For open-source models, all inferences were ran through Together Inference Endpoint.^{33}3h → https://arxiv.org/html/2406.04692v1/https://api.together.ai/playground/chat

3.2 Benchmark Results

In this subsection, we present our evaluation results on three standard benchmarks: AlpacaEval 2.0, MT-Bench, and FLASK. These benchmarks were chosen to comprehensively assess the performance of our approach and compare with the state-of-the-art LLMs.

Table 2: Results on AlpacaEval 2.0 and MT-Bench. For AlpacaEval 2.0, MoA and MoA-Lite correspond to the 6 proposer with 3 layers and with 2 layer respectively. MoA w/ GPT-4o corresponds to using GPT-4o as the final aggregator in MoA. We ran our experiments three times and reported the average scores along with the standard deviation. † denotes our replication of the AlpacaEval results. We ran all the MT-Bench scores ourselves to get turn-based scores.

(a) AlpacaEval 2.0

(b) MT-Bench.

AlpacaEval 2.0

We conducted comparisons against leading models such as GPT-4 and other state-of-the-art open-source models. The detailed results are presented in Table˜2(a) → https://arxiv.org/html/2406.04692v1#S3.T2.st1 where our MoA methodology achieved top positions on the AlpacaEval 2.0 leaderboard, demonstrating a remarkable 8.2% absolute improvement over the previous top model, GPT-4o. Moreover, it is particularly noteworthy that our model outperformed GPT-4o using solely open-source models, achieving a margin of 7.6% absolute improvement from 57.5% (GPT-4o) to 65.1% (MoA). Our MoA-Lite setup uses less layers and being more cost-effective. Even with this lighter approach, we still outperform the best model by 1.8%, improving from 57.5% (GPT-4o) to 59.3% (MoA-Lite). This further highlights the effectiveness of our method in leveraging open-source models capabilities with varying compute budget to their fullest potential.

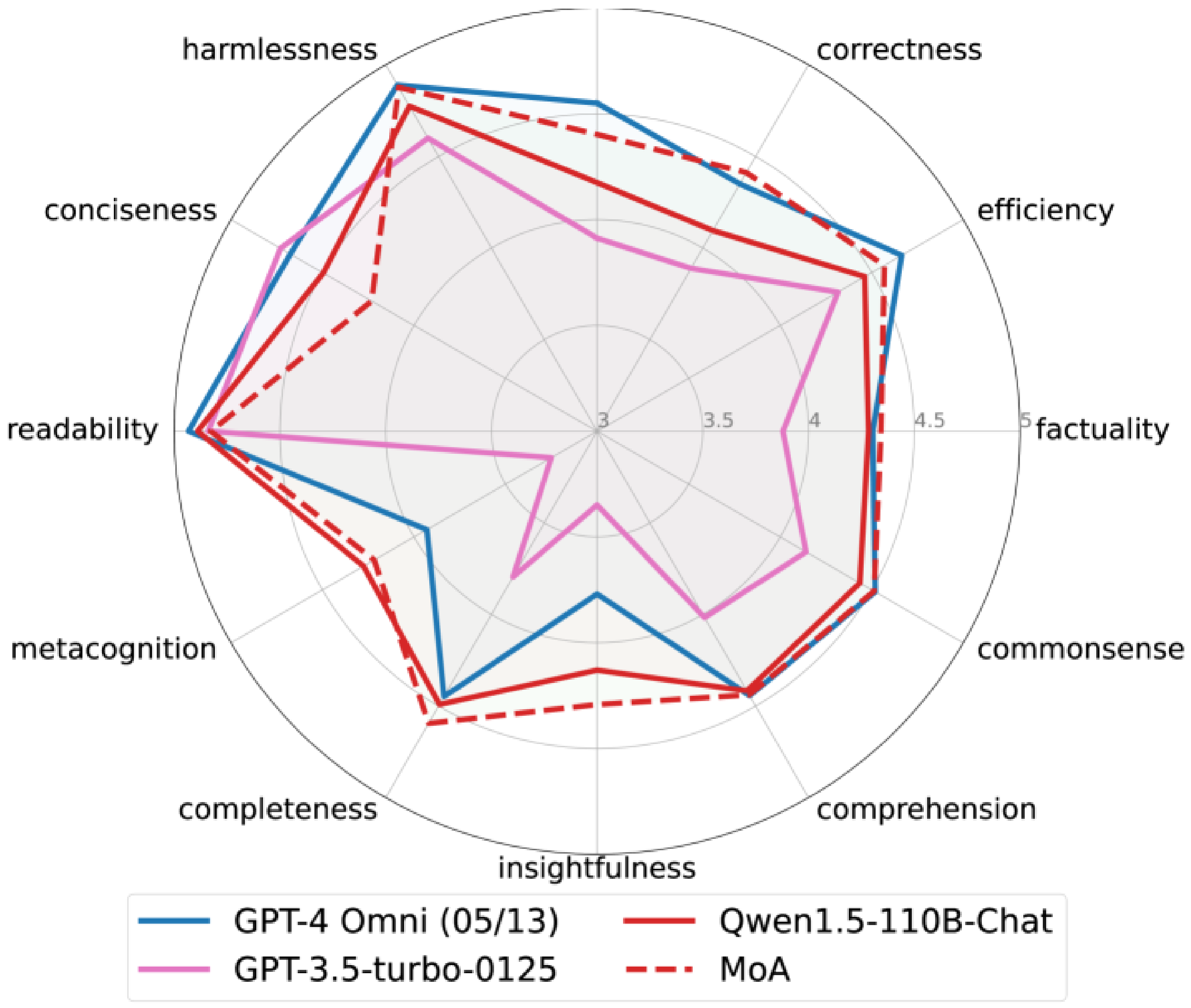


Figure 3: Results on FLASK where we use the 6 proposer MoA setup and Qwen1.5-110B-Chat is the aggregator.

MT-Bench

Though improvements over individual models on the MT-Bench are relatively incremental, this is understandable given that current models already perform exceptionally well on this benchmark, as a single model alone can achieve scores greater than 9 out of 10. Despite the marginal enhancements, our approach still secures the top position on the leaderboard. This demonstrates that even with already highly optimized benchmarks, our method can push the boundaries further, maintaining the leadership.

FLASK

FLASK provides fine-grained evaluation of models. Among those metrics, MoA excels in several key aspects. Specifically, our methodology shows significant improvement in robustness, correctness, efficiency, factuality, commonsense, insightfulness, completeness, compared to the single model score of the aggregator, Qwen-110B-Chat. Additionally, MoA also outperforms GPT-4 Omni in terms of correctness, factuality, insightfulness, completeness, and metacognition. One metric where MoA did not do as well was conciseness; the model produced outputs that were marginally more verbose.

3.3 What Makes Mixture-of-Agents Work Well?

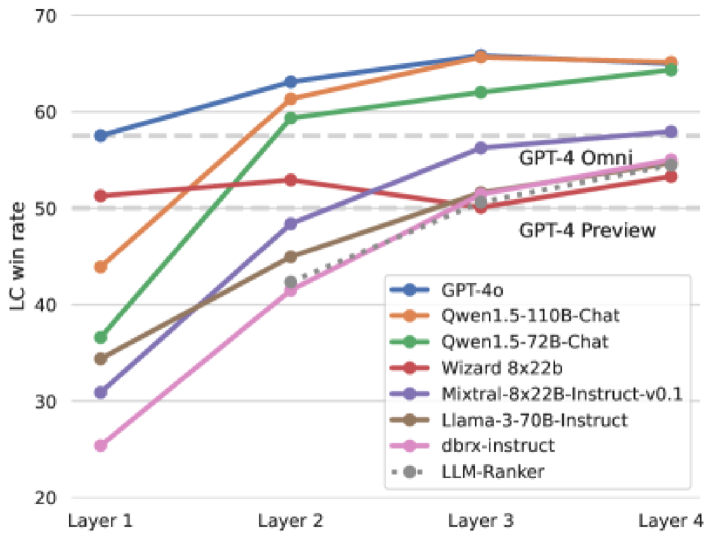
In this subsection, we conduct experiments that provide us better understandings of the internal mechanism of Mixture-of-Agents. We summarize key insights below.

Mixture-of-Agents significantly outperforms LLM rankers.

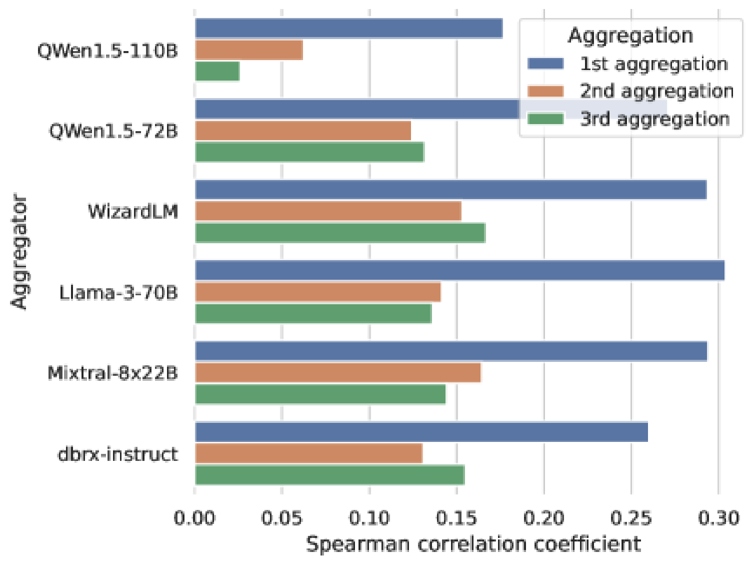
First, we compare Mixture-of-Agents with an LLM-based ranker which uses the aggregator model to select one of the answers that are generated by the proposers, instead of generating a new output. The results are shown in Figure˜4 → https://arxiv.org/html/2406.04692v1#S3.F4, where we can observe that the MoA approach significantly outperforms an LLM-ranker baseline. The fact that MoA outperforms the ranking approach suggests that the aggregator does not simply select one of the generated answers by the proposers, but potentially performs sophisticated aggregation over all proposed generations.

MoA tends to incorporate the best proposed answers.

We also compare the aggregator's response with the proposers' responses via similarity scores such as BLEU *(Papineni et al., 2002 → https://arxiv.org/html/2406.04692v1#bib.bib22)* which reflects n-gram overlaps. Within each sample, given n proposed answers by the proposers, we calculate the the Spearman's rank correlation coefficient between n similar scores and n preference scores determined by the GPT-4 based evaluator. The results in Figure˜4 → https://arxiv.org/html/2406.04692v1#S3.F4 indeed confirms a positive correlation between the win rate and the BLEU score. We also provide results with Levenshtein similarity *(RapidFuzz, 2023 → https://arxiv.org/html/2406.04692v1#bib.bib23)* or TF-IDF as opposed to BLEU scores in Appendix˜A → https://arxiv.org/html/2406.04692v1#A1. where both alternative approaches for textual similarities also yield positive correlation with the preference scores.



Refer to caption



Refer to caption

Figure 4: (a) LC win rate on AlpacaEval 2.0 with different aggregators in the 6-model Mixture-of-Agents setup. All the curves use the same 6 proposer agents; they only differ in the choice of the final aggregator. The LLM ranker uses Qwen1.5-110B-Chat model with a prompt format in Appendix Table˜5 → https://arxiv.org/html/2406.04692v1#A2.T5. The GPT-4o model is only used to aggregate the output for the purpose of evaluation and does not participate as a proposer towards the next layer. (b) Spearman correlation between BLEU scores (calculated using 3-gram, 4-gram, and 5-gram metrics) and win rate of the proposed outputs.

Effect of model diversity and the number of proposers.

We analyze how the number of proposals affect the final output quality by varying n, the number of proposers in each layer. We show the results in Table˜4 → https://arxiv.org/html/2406.04692v1#S3.T4 where we find that scores increases monotonically with n, reflecting the benefits of having more auxiliary information. In addition, we also quantify the impact of using a diverse set of LLMs as proposers. For each n, we compare two settings: "single-proposer" where the n responses are generated by the same LLM with a temperature of 0.7; and "multiple-proposer" where each response is generated by a different LLMs. Overall, using multiple different LLMs consistently yielded better results. Both results suggest that having a larger number of diverse LLM agents in each MoA layer can improve performance. Further scaling the width of MoA is a promising direction of future investigation.

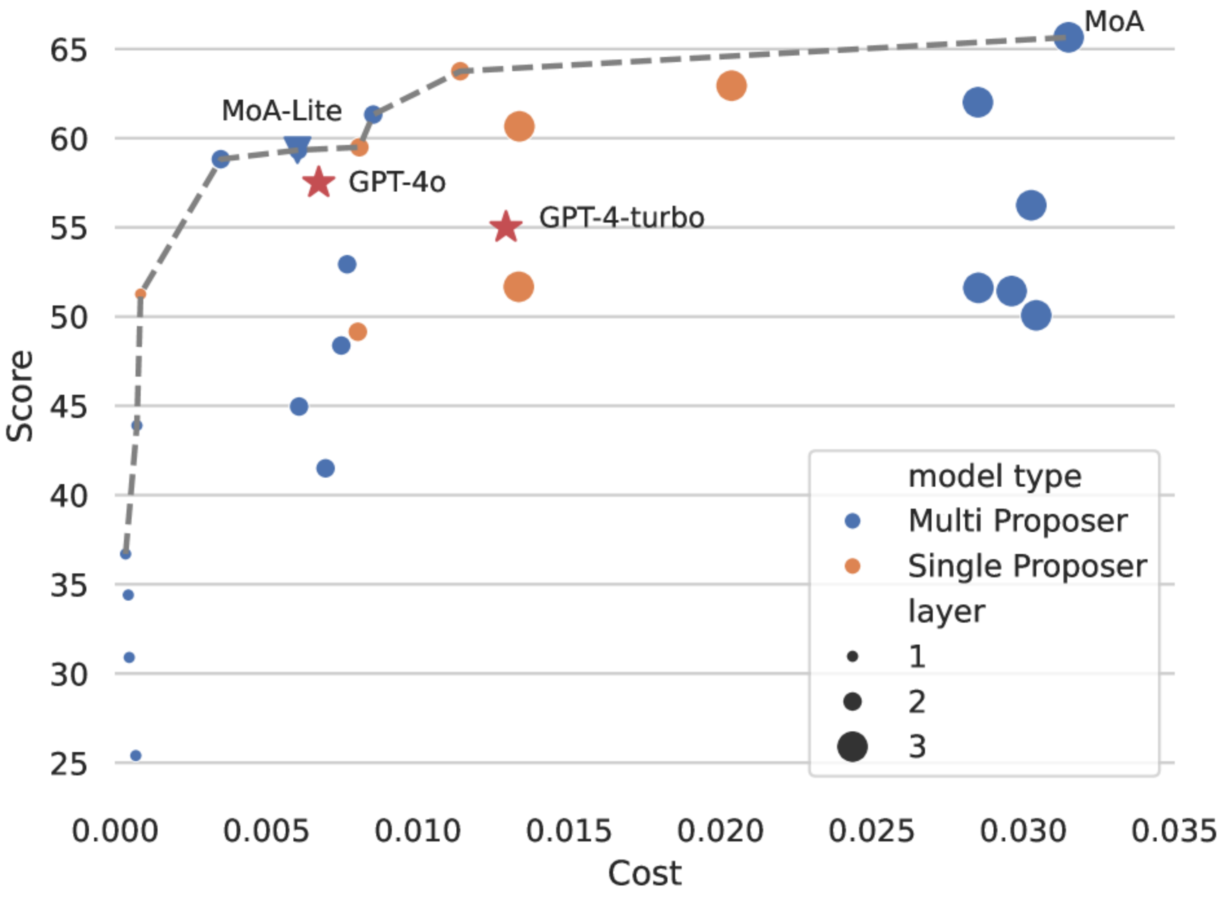
Specialization of models in the Mixture-of-Agent ecosystem.

We also conducted experiments to determine which models excel in specific roles. Specifically, Table˜4 → https://arxiv.org/html/2406.04692v1#S3.T4 shows that GPT-4o, Qwen, LLaMA-3 emerged as a versatile model effective in both assisting and aggregating tasks. In contrast, WizardLM demonstrated excellent performance as an proposer model but struggled to maintain its effectiveness in aggregating responses from other models.

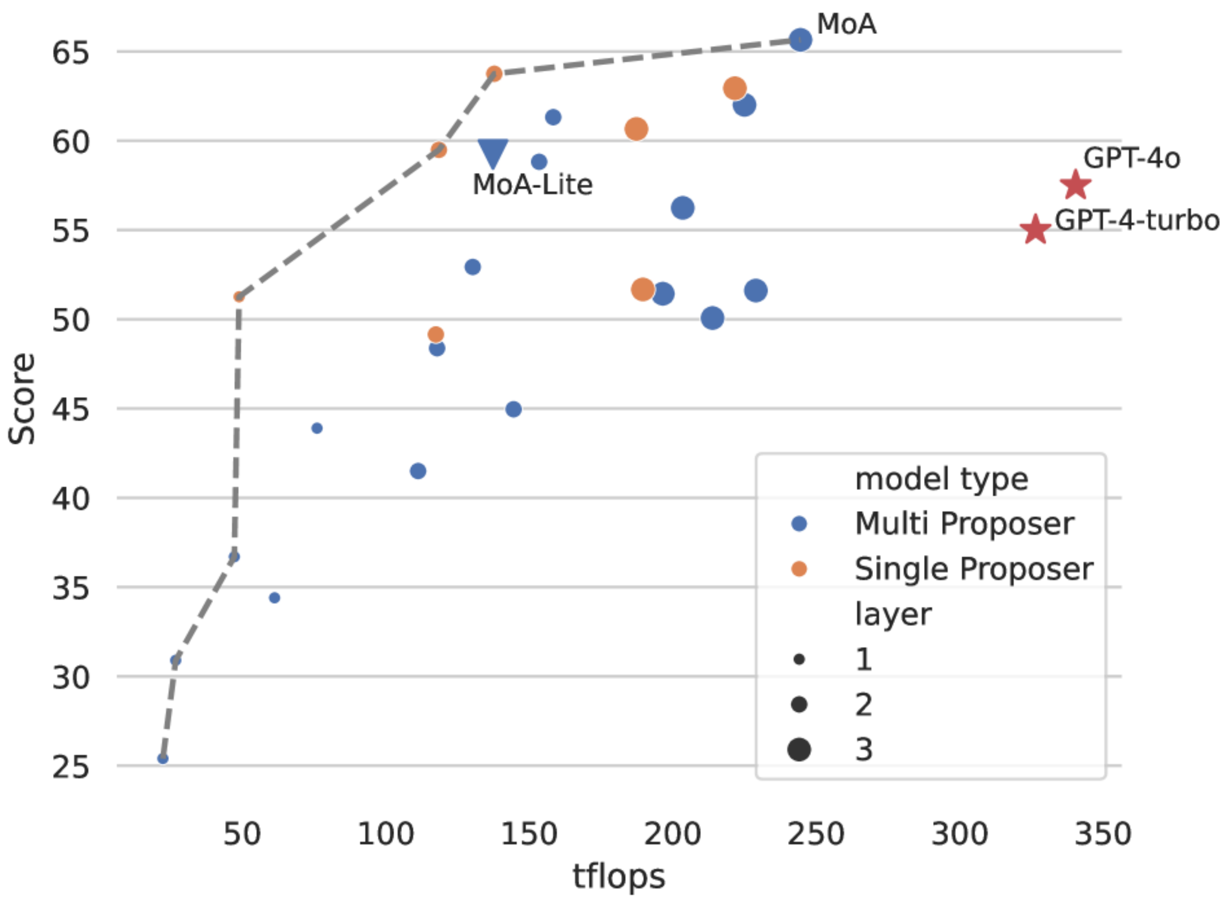
Table 3: Effects of the number of proposer models on AlpacaEval 2.0. We denote n as either the number of agents in an MoA layer or the number of proposed outputs in the single-proposer setting. We use Qwen1.5-110B-Chat as the aggregator and use 2 MoA layers for all settings in this table.

Table 4: Impact of different models serving as proposers vs aggregators. When evaluating different aggregators, all six models serve as proposers; when evaluating proposers, Qwen1.5-110B-Chat serves as the aggregator. We use 2 MoA layers in this table.

3.4 Budget and Token Analysis



(a) LC win rate vs. cost



(b) LC win rate vs. tflops

Figure 5: (a) Performance trade-off versus cost. (b) Performance trade-off versus the number of tera floating operations (tflops), which we use as a proxy for latency. Note that we calculate the sum over layers of the max number of tflops among proposers in each MoA layer as multiple proposers can run in parallel. Our plots illustrate a Pareto frontier where we can choose a model progressively higher score with the lowest cost for such level of performance. We show that the Mixture-of-Agents approach lie on this Pareto front, as opposed to GPT-4 Turbo and GPT-4o which are not cost optimal and is more expensive compared to MoA approaches of the same LC win rate. Single Proposer: uses the same model to generate multiple responses in each MoA layer; Multi Proposer: uses different models in each MoA layer. The actual tflops of GPT-4 is unknown, so we use the rumored size from the community of an 8x220B architecture.

To understand the relationship between budget, token usage, and LC win rates, we conducted a budget and token analysis. Figure˜5(a) → https://arxiv.org/html/2406.04692v1#S3.F5.sf1 and Figure˜5(b) → https://arxiv.org/html/2406.04692v1#S3.F5.sf2 illustrate these relationships.

Cost Effectiveness

In Figure˜5(a) → https://arxiv.org/html/2406.04692v1#S3.F5.sf1, we plot the LC win rate against the average inference cost for each instance in the AplacaEval 2.0 benchmark. The cost is calculated based on pricing information available from API provider websites.[44]4For open-source models, we calculate the price using data from https://api.together.ai/models; for OpenAI models, we use pricing details from https://openai.com/api/pricing/. Pricing data was retrieved as of May 22, 2024. This helps identify cost-effective models that achieve high performance without incurring excessive expenses. The chart reveals a Pareto front where certain models strike an optimal balance between cost and performance. Models closer to this Pareto front are more desirable as they provide better monetary value by delivering high LC win rates at lower costs. Specifically, if we prioritize the quality, MoA is the best configuration. However, if we want to strike a good balance between quality and cost, MoA-Lite can match GPT-4o's cost while achieving higher level of quality. Notably, it outperforms GPT-4 Turbo by approximately 4% while being more than twice as cost-effective.

Tflops Consumption

Figure˜5(b) → https://arxiv.org/html/2406.04692v1#S3.F5.sf2 depicts the relationship between LC win rate and the number of tflops. Here we use the number of tflops as a proxy for latency since latency can vary depending on the inference systems. This analysis is crucial for understanding how different models manage their budgets while maintaining or improving performance levels. Similar to the cost efficiency analysis, a Pareto front can be observed here as well. Models on this front effectively utilize their computational resource to maximize their LC win rate.

4 Related Work

4.1 LLM Reasoning

In order to improve generation quality of LLMs, recent researches have experienced great progresses in optimizing LLMs to various downstream tasks through prompt engineering. Chain of Thought (CoT) *(Wei et al., 2022 → https://arxiv.org/html/2406.04692v1#bib.bib35; Kojima et al., 2022 → https://arxiv.org/html/2406.04692v1#bib.bib16)* prompting techniques represent a linear problem-solving approach where each step builds upon the previous one. *Fu et al. (2022 → https://arxiv.org/html/2406.04692v1#bib.bib10)* applied CoT to multi-step reasoning tasks. To automate CoT prompting, Auto-CoT *(Zhang et al., 2022b → https://arxiv.org/html/2406.04692v1#bib.bib43)* constructs demonstrations by sampling diverse questions and generating reasoning chains. Active-Prompt *(Diao et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib7)* focuses on selecting the most uncertain questions for task-specific annotations. PS Prompt *(Wang et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib32)* decomposes tasks into subtasks. Tree-of-Thought (ToT) *(Yao et al., 2023a → https://arxiv.org/html/2406.04692v1#bib.bib38)* expands on the reasoning process by considering multiple paths of reasoning and self-evaluating choices. Effective Graph-of-Thought *(Yao et al., 2023b → https://arxiv.org/html/2406.04692v1#bib.bib39)* frames thoughts as graphs. Natural Program prompting *(Ling et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib18)* is proposed for better solving deductive reasoning tasks. And re-reading prompt *(Xu et al., 2023b → https://arxiv.org/html/2406.04692v1#bib.bib37)* revisits question information embedded within input prompts.

4.2 Model Ensemble

A straightforward solution to leverage the strengths of multiple models is reranking outputs from different models. For instance, *Jiang et al. (2023 → https://arxiv.org/html/2406.04692v1#bib.bib15)* introduce PairRanker, which performs pairwise comparisons on candidate outputs to select the best one, showing improvements on a self-constructed instruction dataset. To address the substantial computational costs associated with multi-LLM inference, other studies have explored training a router that predicts the best-performing model from a fixed set of LLMs for a given input *(Wang et al., 2024a → https://arxiv.org/html/2406.04692v1#bib.bib31; Shnitzer et al., 2024 → https://arxiv.org/html/2406.04692v1#bib.bib26; Lu et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib19)*. Additionally, FrugalGPT *(Chen et al., 2023b → https://arxiv.org/html/2406.04692v1#bib.bib5)* proposed reducing the cost of using LLMs by employing different models in a cascading manner. In order to better leverage the responses of multiple models, *Jiang et al. (2023 → https://arxiv.org/html/2406.04692v1#bib.bib15)* trained a GenFuser, a model that was trained to generate an improved response to capitalize on the strengths of multiple candidates. *Huang et al. (2024 → https://arxiv.org/html/2406.04692v1#bib.bib13)* proposed to fuse the outputs of different models by averaging their output probability distributions.

Another line of work is multi-agent collaboration. Several studies explore using multiple large language models as agents that collectively discuss and reason through given problems interactively. *Du et al. (2023 → https://arxiv.org/html/2406.04692v1#bib.bib8)* establishes a mechanism for symmetric discussions among agents. Around the same time, MAD *(Liang et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib17)* introduces an asymmetric mechanism design, with different roles, i.e., debater and judge. Other similar works include *(Chan et al., 2023 → https://arxiv.org/html/2406.04692v1#bib.bib3)*. Moreover, ReConcile *(Chen et al., 2023a → https://arxiv.org/html/2406.04692v1#bib.bib4)* exemplifies an asymmetric discussion involving weighted voting. To understand discussion more deeply, *Zhang et al. (2023 → https://arxiv.org/html/2406.04692v1#bib.bib41)* aim to explain such collaboration mechanism in a social psychology view. *Wang et al. (2024b → https://arxiv.org/html/2406.04692v1#bib.bib33)* systematically compared multi-agent approaches and found a single agent with a strong prompt including detailed demonstrations can achieve comparable response quality to multi-agent approaches.

5 Conclusion

This paper introduces a Mixture-of-Agents approach aimed at leveraging the capabilities of multiple LLMs via successive stages for iterative collaboration. Our method harnesses the collective strengths of agents in the Mixture-of-Agents family, and can significantly improve upon the output quality of each individual model. Empirical evaluations conducted on AlpacaEval 2.0, MT-Bench, and FLASK demonstrated substantial improvements in response quality, with our approach achieving the LC win rate up to 65%. These findings validate our hypothesis that integrating diverse perspectives from various models can lead to superior performance compared to relying on a single model alone. In addition, we provide insights into improving the design of MoA; systematic optimization of MoA architecture is an interesting direction for future work.

Limitations.

Our proposed method requires iterative aggregation of model responses, which means the model cannot decide the first token until the last MoA layer is reached. This potentially results in a high Time to First Token (TTFT), which can negatively impact user experience. To mitigate this issue, we can limit the number of MoA layers, as the first response aggregation has the most significant boost on generation quality. Future work could explore chunk-wise aggregation instead of aggregating entire responses at once, which can reduce TTFT while maintaining response quality.

Broader Impact.

This study holds the potential to enhance the effectiveness of LLM-driven chat assistants, thereby making AI more accessible. Moreover, since the intermediate outputs that are expressed in natural language, MoA presented improves the interpretability of models. This enhanced interpretability facilitates better alignment with human reasoning.

References

- Bai et al. (2023) Bai, J., Bai, S., Chu, Y., Cui, Z., Dang, K., Deng, X., Fan, Y., Ge, W., Han, Y., Huang, F., et al. Qwen technical report. *arXiv preprint arXiv:2309.16609*, 2023.

- Brown et al. (2020) Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.
- Chan et al. (2023) Chan, C.-M., Chen, W., Su, Y., Yu, J., Xue, W., Zhang, S., Fu, J., and Liu, Z. Chateval: Towards better llm-based evaluators through multi-agent debate. *arXiv preprint arXiv:2308.07201*, 2023.
- Chen et al. (2023a) Chen, J. C.-Y., Saha, S., and Bansal, M. Reconcile: Round-table conference improves reasoning via consensus among diverse llms. *arXiv preprint arXiv:2309.13007*, 2023a.
- Chen et al. (2023b) Chen, L., Zaharia, M., and Zou, J. Frugalgpt: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*, 2023b.
- Chowdhery et al. (2022) Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*, 2022.
- Diao et al. (2023) Diao, S., Wang, P., Lin, Y., and Zhang, T. Active prompting with chain-of-thought for large language models. *arXiv preprint arXiv:2302.12246*, 2023.
- Du et al. (2023) Du, Y., Li, S., Torralba, A., Tenenbaum, J. B., and Mordatch, I. Improving factuality and reasoning in language models through multiagent debate. *arXiv preprint arXiv:2305.14325*, 2023.
- Dubois et al. (2024) Dubois, Y., Galambosi, B., Liang, P., and Hashimoto, T. B. Length-controlled alpacaeval: A simple way to debias automatic evaluators. *arXiv preprint arXiv:2404.04475*, 2024.
- Fu et al. (2022) Fu, Y., Peng, H., Sabharwal, A., Clark, P., and Khot, T. Complexity-based prompting for multi-step reasoning. *arXiv preprint arXiv:2210.00720*, 2022.
- Guo et al. (2024) Guo, D., Zhu, Q., Yang, D., Xie, Z., Dong, K., Zhang, W., Chen, G., Bi, X., Wu, Y., Li, Y., et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.
- Hendrycks et al. (2021) Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. *arXiv preprint arXiv:2103.03874*, 2021.
- Huang et al. (2024) Huang, Y., Feng, X., Li, B., Xiang, Y., Wang, H., Qin, B., and Liu, T. Enabling ensemble learning for heterogeneous large language models with deep parallel collaboration. *arXiv preprint arXiv:2404.12715*, 2024.
- Jiang et al. (2024) Jiang, A. Q., Sablayrolles, A., Roux, A., Mensch, A., Savary, B., Bamford, C., Chaplot, D. S., de Las Casas, D., Hanna, E. B., Bressand, F., Lengyel, G., Bour, G., Lample, G., Lavaud, L. R., Saulnier, L., Lachaux, M., Stock, P., Subramanian, S., Yang, S., Antoniak, S., Scao, T. L., Gervet, T., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. Mixtral of experts. *CoRR*, abs/2401.04088, 2024. doi: 10.48550/ARXIV.2401.04088. URL https://doi.org/10.48550/arXiv.2401.04088.
- Jiang et al. (2023) Jiang, D., Ren, X., and Lin, B. Y. LLM-blender: Ensembling large language models with pairwise ranking and generative fusion. In Rogers, A., Boyd-Graber, J., and Okazaki, N. (eds.), *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14165–14178, Toronto, Canada, July 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.acl-long.792. URL https://aclanthology.org/2023.acl-long.792.
- Kojima et al. (2022) Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., and Iwasawa, Y. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213, 2022.
- Liang et al. (2023) Liang, T., He, Z., Jiao, W., Wang, X., Wang, Y., Wang, R., Yang, Y., Tu, Z., and Shi, S. Encouraging divergent thinking in large language models through multi-agent debate. *arXiv preprint arXiv:2305.19118*, 2023.

- Ling et al. (2023) Ling, Z., Fang, Y., Li, X., Huang, Z., Lee, M., Memisevic, R., and Su, H. Deductive verification of chain-of-thought reasoning. *arXiv preprint arXiv:2306.03872*, 2023.
- Lu et al. (2023) Lu, K., Yuan, H., Lin, R., Lin, J., Yuan, Z., Zhou, C., and Zhou, J. Routing to the expert: Efficient reward-guided ensemble of large language models, 2023.
- OpenAI (2023) OpenAI. Gpt-4 technical report, 2023.
- Ouyang et al. (2022) Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.
- Papineni et al. (2002) Papineni, K., Roukos, S., Ward, T., and Zhu, W. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*, pp. 311–318. ACL, 2002. doi: 10.3115/1073083.1073135. URL https://aclanthology.org/P02-1040/.
- RapidFuzz (2023) RapidFuzz. python-levenshtein by rapidfuzz. https://github.com/rapidfuzz/python-Levenshtein, 2023.
- Roziere et al. (2023) Roziere, B., Gehring, J., Gloeckle, F., Sootla, S., Gat, I., Tan, X. E., Adi, Y., Liu, J., Remez, T., Rapin, J., et al. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*, 2023.
- Shazeer et al. (2017) Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. *arXiv preprint arXiv:1701.06538*, 2017.
- Shnitzer et al. (2024) Shnitzer, T., Ou, A., Silva, M., Soule, K., Sun, Y., Solomon, J., Thompson, N., and Yurochkin, M. Large language model routing with benchmark datasets, 2024. URL https://openreview.net/forum?id=LyNsMNNLjY.
- Team et al. (2023) Team, G., Anil, R., Borgeaud, S., Wu, Y., Alayrac, J.-B., Yu, J., Soricut, R., Schalkwyk, J., Dai, A. M., Hauth, A., et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023.
- The Mosaic Research Team (2024) The Mosaic Research Team. Introducing dbrx: A new state-of-the-art open llm. 2024. URL https://www.databricks.com/blog/introducing-dbrx-new-state-art-open-llm.
- Touvron et al. (2023a) Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.
- Touvron et al. (2023b) Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.
- Wang et al. (2024a) Wang, H., Polo, F. M., Sun, Y., Kundu, S., Xing, E., and Yurochkin, M. Fusing models with complementary expertise. In *The Twelfth International Conference on Learning Representations*, 2024a. URL https://openreview.net/forum?id=PhMrGCMIRL.
- Wang et al. (2023) Wang, L., Xu, W., Lan, Y., Hu, Z., Lan, Y., Lee, R. K.-W., and Lim, E.-P. Plan-and-solve prompting: Improving zero-shot chain-of-thought reasoning by large language models. *arXiv preprint arXiv:2305.04091*, 2023.
- Wang et al. (2024b) Wang, Q., Wang, Z., Su, Y., Tong, H., and Song, Y. Rethinking the bounds of llm reasoning: Are multi-agent discussions the key? *arXiv preprint arXiv:2402.18272*, 2024b.
- Wang et al. (2022) Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., Chowdhery, A., and Zhou, D. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.
- Wei et al. (2022) Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., Le, Q. V., Zhou, D., et al. Chain-of-thought prompting elicits reasoning in large language

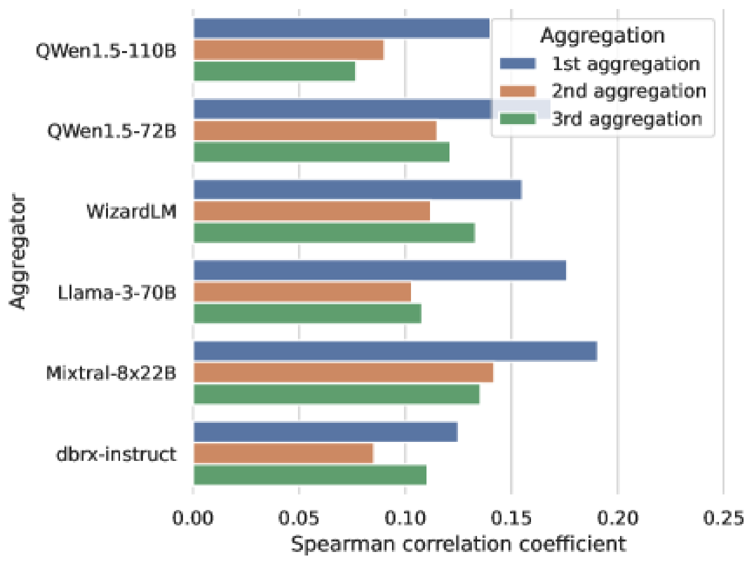
models. *Advances in Neural Information Processing Systems*, 35:24824–24837, 2022.

- Xu et al. (2023a) Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C., and Jiang, D. Wizardlm: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023a.
- Xu et al. (2023b) Xu, X., Tao, C., Shen, T., Xu, C., Xu, H., Long, G., and Lou, J.-g. Re-reading improves reasoning in language models. *arXiv preprint arXiv:2309.06275*, 2023b.
- Yao et al. (2023a) Yao, S., Yu, D., Zhao, J., Shafran, I., Griffiths, T. L., Cao, Y., and Narasimhan, K. Tree of thoughts: Deliberate problem solving with large language models. *arXiv preprint arXiv:2305.10601*, 2023a.
- Yao et al. (2023b) Yao, Y., Li, Z., and Zhao, H. Beyond chain-of-thought, effective graph-of-thought reasoning in large language models. *arXiv preprint arXiv:2305.16582*, 2023b.
- Ye et al. (2023) Ye, S., Kim, D., Kim, S., Hwang, H., Kim, S., Jo, Y., Thorne, J., Kim, J., and Seo, M. Flask: Fine-grained language model evaluation based on alignment skill sets. *arXiv preprint arXiv:2307.10928*, 2023.
- Zhang et al. (2023) Zhang, J., Xu, X., and Deng, S. Exploring collaboration mechanisms for llm agents: A social psychology view. *arXiv preprint arXiv:2310.02124*, 2023.
- Zhang et al. (2022a) Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., et al. Opt: Open pre-trained transformer language models. *arXiv e-prints*, pp. arXiv–2205, 2022a.
- Zhang et al. (2022b) Zhang, Z., Zhang, A., Li, M., and Smola, A. Automatic chain of thought prompting in large language models. *arXiv preprint arXiv:2210.03493*, 2022b.
- Zheng et al. (2023) Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging llm-as-a-judge with mt-bench and chatbot arena. *arXiv preprint arXiv:2306.05685*, 2023.

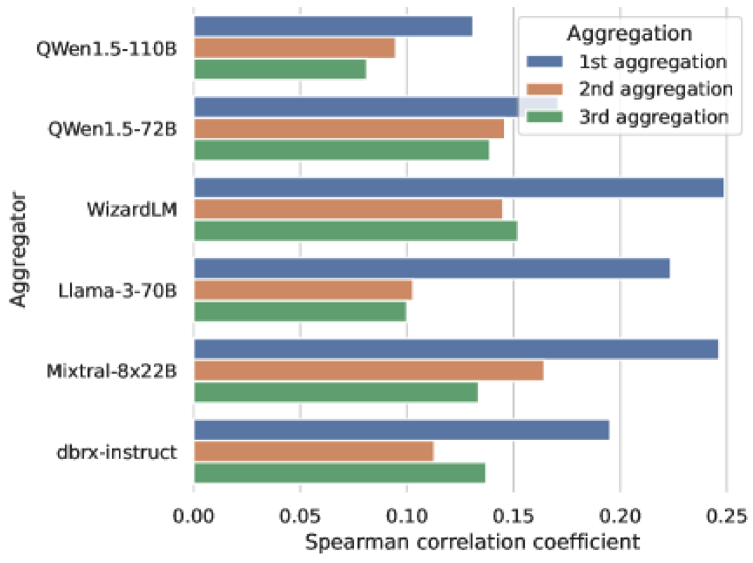
Supplementary Material

Appendix A Spearman Correlation using Different Similarity Functions

We present results using TF-IDF-based similarity and Levenshtein similarity when calculating the Spearman correlation. Specifically, within each sample of n proposed answers, we calculate Spearman correlation coefficient between the n similarity scores and the n preference scores determined by the GPT-4-based evaluator. As shown in Figure˜6 → https://arxiv.org/html/2406.04692v1#A1.F6, there is indeed a positive correlation between win rate and both TF-IDF similarity and Levenshtein similarity.



Refer to caption



Refer to caption

Figure 6: (a) Spearman Correlation using TF-IDF similarity; (b) Spearman Correlation using Levenshtein similarity.

Appendix B LLM Ranker

This section introduces the setup of the LLM-Ranker used in this paper. The LLM-Ranker is designed to evaluate and rank the best output generated by some LLMs. Table˜5 → https://arxiv.org/html/2406.04692v1#A2.T5 presents the template for prompting the model during these evaluations. We use this LLM-Ranker to pick the best answer among and use AlpacaEval evaluator to evaluate the best ranked answer.

Table 5: Prompt for ranking with LLMs

You are a highly efficient assistant, who evaluates and selects the best large language model (LLMs) based on the quality of their responses to a given instruction. This process will be used to create a leaderboard reflecting the most accurate and human-preferred answers.

I require a leaderboard for various large language models. I'll provide you with prompts given to these models and their corresponding outputs. Your task is to assess these responses, and select the model that produces the best output from a human perspective.

Instruction

```
{
    "instruction": """{instruction}""",
}
```

Model Outputs

Here are the unordered outputs from the models. Each output is associated with a specific model, identified by a unique model identifier.

```
{
    {
        "model_identifier": "{identifier_1}",
        "output": """{output_1}"""
    },
    {
        "model_identifier": "{identifier_2}",
        "output": """{output_2}"""
    },
    {
        "model_identifier": "{identifier_3}",
        "output": """{output_3}"""
    },
    {
        "model_identifier": "{identifier_4}",
        "output": """{output_4}"""
    },
    {
        "model_identifier": "{identifier_5}",
        "output": """{output_5}"""
    },
    {
        "model_identifier": "{identifier_6}",
        "output": """{output_6}"""
    }
}
```

Task

Evaluate the models based on the quality and relevance of their outputs, and select the model that generated the best output. Answer by providing the model identifier of the best model. We will use your output as the name of the best model, so make sure your output only contains one of the following model identifiers and nothing else (no quotes, no spaces, no new lines, ...).

Best Model Identifier

Appendix C Case Study

Table 6: Case: Some models produce high quality answers.

Role	Preference	Content
User	-	I want to talk about "Smooth", a song by Rob Thomas.
Qwen1.5-110B-Chat	0.35	… It’s a perfect blend of rock, pop, and Latin music, showcasing Santana’s signature guitar work and Rob Thomas’s soulful vocals …
Qwen1.5-72B-Chat	0.00	…
Llama-3-70B-Instruct	0.00	… it’s a collaboration between Rob Thomas, the lead vocalist of Matchbox Twenty, and Santana, the legendary guitarist …
WizardLM-2-8x22B	0.99	… It spent 12 weeks at the top of the Billboard Hot 100 chart in the United States … capturing the song’s sultry and energetic vibe …
Mixtral-8x22B-Instruct-v0.1	0.00	…
dbrx-instruc	0.00	…
Aggregated (Qwen1.5-110B-Chat)	0.99	"Smooth" is actually a collaboration between Rob Thomas, who is the lead singer of the band Matchbox Twenty, and the renowned Latin rock guitarist Carlos Santana … This iconic song blends rock, pop, and Latin music elements, showcasing Santana’s signature guitar work and Rob Thomas’s distinctive vocals … The song was a colossal success, spending 12 weeks at the top of the Billboard Hot 100 … The music video, set in a vibrant club scene, complements the song’s sultry vibe …

Table 7: Case: all proposed responses are not good enough.

Role	Preference	Content
User	-	How do you become an author?
Qwen1.5-110B-Chat	0.00	Becoming an author involves several steps and … 3. **Choose your niche**:… 8. **Understand the publishing industry**:… 10. **Self-publishing options**:…
Qwen1.5-72B-Chat	0.00	…
Llama-3-70B-Instruct	0.16	Becoming an author can be a rewarding and fulfilling career… 1. **Develop a love for reading and writing**:… 10. **Market and promote your work**:… 11. **Continuously improve**:…
WizardLM-2-8x22B	0.03	1. … 4. Finish your work:…
Mixtral-8x22B-Instruct-v0.1	0.00	…
dbrx-instruc	0.00	…
Aggregated (Qwen1.5-110B-Chat)	0.33	Becoming an author involves a dedicated journey… 1. **Cultivate a Love for Writing and Reading**:… 4. **Choose Your Niche**:… 5. **Finish Your Work**:… 10. **Self-Publishing**:… 11. **Marketing and Promotion**:… 12. **Continuous Learning and Writing**:…

We present a case study in this section. Due to the length of the responses generated by all models, we will only show selected fragments for brevity. To illustrate how the aggregator synthesizes the response, we underlined similar expressions between the proposed responses and the aggregated response in different colors. We omit the content that all proposed responses have mentioned.

Table˜6 → https://arxiv.org/html/2406.04692v1#A3.T6 showcases the responses generated by different proposers. The aggregated response generated by Qwen1.5-110B-Chat reflects a high preference for its own content but also incorporates key points from Llama-3-70B-Instruct and WizardLM 8x22B. Notably, GPT-4's preference score for WizardLM 8x22B's response is 0.99, and the final aggregated answer also achieves a preference score of 0.99.

Meanwhile, Table˜7 → https://arxiv.org/html/2406.04692v1#A3.T7 presents another case where none of the proposed responses achieve a high GPT-4 preference score. Despite this, the aggregator successfully identifies and incorporates the strong points from these responses, achieving a preference score of 0.33.

Appendix D MATH Task

Here, we demonstrate that our approach is applicable to reasoning tasks, such as those in the MATH dataset *Hendrycks et al. (2021 → https://arxiv.org/html/2406.04692v1#bib.bib12)*. The results are presented in Table˜8 → https://arxiv.org/html/2406.04692v1#A4.T8, where we show that our method consistently enhances accuracy by a significant margin. This indicates that our approach is also effective for this type of task. Notably, our method is complementary to existing reasoning techniques such as Chain of Thought *Wei et al. (2022 → https://arxiv.org/html/2406.04692v1#bib.bib35)* and Self-consistency *Wang et al. (2022 → https://arxiv.org/html/2406.04692v1#bib.bib34)*.

Table 8: Results on the MATH task. We evaluate different aggregators, with all six models serving as proposers in each MoA layer.

Documentation - Docling

Source: https://docling-project.github.io/docling/

PDF
PPTX
DOCX
HTML
docling
Docling {DOC}
JSON, MD, Figures ...
{...}
#...
Chunking, LlamaIndex, LangChain
Your GenAI App

1 GITHUB TRENDING **#1 Repository Of The D**

DS4SD%2Fdocling | Trendshift

arXiv 2408.09869

arXiv

pypi v2.66.0

PyPI version

python 3.9 | 3.10 | 3.11 | 3.12 | 3.13 | 3.14

PyPI - Python Version

uv

uv

Ruff

Ruff

Pydantic v2

Pydantic v2

pre-commit enabled

pre-commit

license MIT

License MIT

downloads/month 2M

PyPI Downloads

Apify 341 users / 1.3K runs

Docling Actor

Chat

Chat with Dosu

chat 30 online

Discord

openssf best practices silver

OpenSSF Best Practices

LF AI & Data

LF AI & Data

Docling simplifies document processing, parsing diverse formats — including advanced PDF understanding — and providing seamless integrations with the gen AI ecosystem.

Getting started

🐣 Ready to kick off your Docling journey? Let's dive right into it!

Features

- 🗂️ Parsing of multiple document formats → https://docling-project.github.io/docling/usage/supported_formats/ incl. PDF, DOCX, PPTX, XLSX, HTML, WAV, MP3, VTT, images (PNG, TIFF, JPEG, ...), and more
- 📑 Advanced PDF understanding incl. page layout, reading order, table structure, code, formulas, image classification, and more
- 🧬 Unified, expressive DoclingDocument → https://docling-project.github.io/docling/concepts/docling_document/ representation format
- ↪️ Various export formats → https://docling-project.github.io/docling/usage/supported_formats/ and options, including Markdown, HTML, DocTags → https://arxiv.org/abs/2503.11576 and lossless JSON
- 🔒 Local execution capabilities for sensitive data and air-gapped environments
- 🤖 Plug-and-play integrations → https://docling-project.github.io/docling/integrations/ incl. LangChain, LlamaIndex, Crew AI & Haystack for agentic AI
- 🔍 Extensive OCR support for scanned PDFs and images
- 👓 Support of several Visual Language Models (GraniteDocling → https://huggingface.co/ibm-granite/granite-docling-258M)
- 🎙️ Support for Audio with Automatic Speech Recognition (ASR) models
- 🔌 Connect to any agent using the Docling MCP → https://docling-project.github.io/docling/usage/mcp/ server
- 💻 Simple and convenient CLI

What's new

- 📤 Structured [information extraction][extraction] [🧪 beta]
- 📑 New layout model (**Heron**) by default, for faster PDF parsing
- 🔌 MCP server → https://docling-project.github.io/docling/usage/mcp/ for agentic applications
- 💬 Parsing of Web Video Text Tracks (WebVTT) files

Coming soon

- 📝 Metadata extraction, including title, authors, references & language
- 📝 Chart understanding (Barchart, Piechart, LinePlot, etc)
- 📝 Complex chemistry understanding (Molecular structures)

What's next

🚀 The journey has just begun! Join us and become a part of the growing Docling community.

- GitHub → https://github.com/docling-project/docling

- Discord → https://docling.ai/discord

- LinkedIn → https://linkedin.com/company/docling/

Live assistant

Do you want to leverage the power of AI and get live support on Docling? Try out the Chat with Dosu → https://app.dosu.dev/097760a8-135e-4789-8234-90c8837d7f1c/ask?utm_source=github functionalities provided by our friends at Dosu → https://dosu.dev/.

Chat

Chat with Dosu

LF AI & Data

Docling is hosted as a project in the LF AI & Data Foundation → https://lfaidata.foundation/projects/.

IBM ❤ Open Source AI

The project was started by the AI for knowledge team at IBM Research Zurich.

Meshtastic Map

Source: https://meshtastic.liamcottle.net/

How do I add my node to the map?

Your node, or a node that hears your node must uplink to our MQTT server.

Your position packet must be unencrypted, or encrypted with the default key.

If your node has v2.5 firmware or newer, you must enable "OK to MQTT".

Use the MQTT server details below to uplink to this map.

What MQTT server should I use?

- Address: mqtt.meshtastic.liamcottle.net
- Username: uplink
- Password: uplink
- Encryption Enabled: Yes
- JSON Output: No
- TLS Enabled: No

Please note, nodes can only uplink to this server. Downlink is disabled.

We suggest running dedicated nodes for uplinking to the map.

How do I remove my node from the map?

Nodes that have not been heard for 7 days are automatically removed.

Disable position reporting in your node to prevent it coming back.

Use custom encryption keys so the public can't see your position data.

If your node has v2.5 firmware or newer, we ignore packets if "OK to MQTT" is disabled.

To have your node removed now, please contact me → https://liamcottle.com/contact.

How do I see neighbours a node heard?

Open the top right layers panel and enable neighbours.

Some neighbours are from MQTT, this is patched in latest firmware.

Why is my node showing up in the wrong place?

This public map obfuscates your position packets for v2.4 firmware and older.

Nodes on v2.4 and older have their positions obfuscated to 2 decimal places.

Nodes on v2.5 and newer, with "OK to MQTT" enabled, will show positions with your configured precision.

Nodes on v2.5 and newer, with "OK to MQTT" disabled, will not update their positions.

Stop Writing `__init__` Methods

By Glyph Lefkowitz
Source: https://blog.glyph.im/2025/04/stop-writing-init-methods.html

The History

Before dataclasses were added to Python in version 3.7 — in June of 2018 — the `__init__` special method had an important use. If you had a class representing a data structure — for example a `2DCoordinate`, with `x` and `y` attributes — you would want to be able to construct it as `2DCoordinate(x=1, y=2)`, which would require you to add an `__init__` method with `x` and `y` parameters.

The other options available at the time all had pretty bad problems:

1. You could remove `2DCoordinate` from your public API and instead expose a `make_2d_coordinate` function and make it non-importable, but then how would you document your return or parameter types?
2. You could document the `x` and `y` attributes and make the user assign each one themselves, but then `2DCoordinate()` would return an invalid object.
3. You could default your coordinates to 0 with class attributes, and while that would fix the problem with option 2, this would now require all `2DCoordinate` objects to be not just mutable, but mutated at every call site.
4. You could fix the problems with option 1 by adding a new *abstract* class that you could expose in your public API, but this would explode the complexity of every new public class, no matter how simple. To make matters worse, `typing.Protocol` didn't even arrive until Python 3.8, so, in the pre-3.7 world this would condemn you to using concrete inheritance and declaring multiple classes even for the most basic data structure imaginable.

Also, an `__init__` method *that does nothing but assign a few attributes* doesn't have any significant problems, so it is an obvious choice in this case. Given all the problems that I just described with the alternatives, it makes sense that it became the obvious *default* choice, in most cases.

However, by accepting "define a custom `__init__`" as the *default* way to allow users to create your objects, we make a habit of beginning *every* class with a pile of *arbitrary code* that gets executed every time it is instantiated.

Wherever there is arbitrary code, there are arbitrary problems.

The Problems

Let's consider a data structure more complex than one that simply holds a couple of attributes. We will create one that represents a reference to some I/O in the external world: a `FileReader`.

Of course Python has its own open-file object abstraction → https://docs.python.org/3.13/library/io.html#io.FileIO, but I will be ignoring that for the purposes of the example.

Let's assume a world where we have the following functions, in an imaginary `fileio` module:

- `open(path: str) -> int`
- `read(fileno: int, length: int)`
- `close(fileno: int)`

Our hypothetical `fileio.open` returns an integer representing a file descriptor[1], `fileio.read` allows us to read `length` bytes from an open file descriptor, and `fileio.close` closes that file descriptor, invalidating it for future use.

With the habit that we have built from writing thousands of `__init__` methods, we might want to write our `FileReader` class like this:

```
class FileReader:
    def __init__(self, path: str) -> None:
        self._fd = fileio.open(path)
    def read(self, length: int) -> bytes:
        return fileio.read(self._fd, length)
    def close(self) -> None:
        fileio.close(self._fd)
```

For our initial use-case, this is fine. Client code creates a `FileReader` by doing something like `FileReader("./config.json")`, which always creates a `FileReader` that maintains its file descriptor `int` internally as private state. This is as it should be; we don't want user code to see or mess with `_fd`, as that might violate `FileReader`'s invariants. All the necessary work to construct a valid `FileReader` — i.e. the call to `open` — is always taken care of for you by `FileReader.__init__`.

However, additional requirements will creep in, and as they do, `FileReader.__init__` becomes increasingly awkward.

Initially we only care about `fileio.open`, but later, we may have to deal with a library that has its own reasons for managing the call to `fileio.open` by itself, and wants to give us an `int` that we use as our `_fd`, we now have to resort to weird workarounds like:

```
def reader_from_fd(fd: int) -> FileReader:
    fr = object.__new__(FileReader)
    fr._fd = fd
    return fr
```

Now, all those nice properties that we got from trying to force object construction to give us a valid object are gone. `reader_from_fd`'s type signature, which takes a plain `int`, has no way of even suggesting to client code how to ensure that it has passed in the right *kind* of `int`.

Testing is much more of a hassle, because we have to patch in our own copy of `fileio.open` any time we want an instance of a `FileReader` in a test without doing any real-life file I/O, even if we could (for example) share a single file descriptor among many `Fil-eReader` s for testing purposes.

All of this also assumes a `fileio.open` that is *synchronous*. Although for literal file I/O this is more of a hypothetical → https://stackoverflow.com/questions/87892/what-is-the-status-of-posix-asynchronous-i-o-aio concern, there are many types of networked resource which are really only available via an asynchronous (and thus: potentially slow, potentially error-prone) API. If you've ever found yourself wanting to type `async def __init__(self): ...` then you have seen this limitation in practice.

Comprehensively describing *all* the possible problems with this approach would end up being a book-length treatise on a philosophy of object oriented design, so I will sum up by saying that the *cause* of all these problems is the same: we are inextricably linking the act of *creating a data structure* with *whatever side-effects are most often associated* with that data structure. If they are “often” associated with it, then by definition they are not “always” associated with it, and all the cases where they *aren’t* associated become unweildy and potentially broken.

Defining an `__init__` is an anti-pattern, and we need a replacement for it.

The Solutions

I believe this tripartite assemblage of design techniques will address the problems raised above:

- using `dataclass` to define attributes,
- replacing behavior that previously would have previously been in `__init__` with a new classmethod that does the same thing, and
- using precise types to describe what a valid instance looks like.

Using `dataclass` attributes to create an `__init__` for you

To begin, let’s refactor `FileReader` into a `dataclass`. This does get us an `__init__` method, but it *won’t* be one an arbitrary one we define ourselves; it will get the useful constraint enforced on it that it will just assign attributes.

```
@dataclass
class FileReader:
    _fd: int
    def read(self, length: int) -> bytes:
        return fileio.read(self._fd, length)
    def close(self) -> None:
        fileio.close(self._fd)
```

Except... oops. In fixing the problems that we created with our custom `__init__` that calls `fileio.open`, we have re-introduced several problems that it solved:

1. We have removed all the convenience of `FileReader("path")`. Now the user needs to import the low-level `fileio.open` again, making the most common type of construction both more verbose and less discoverable; if we want users to know how to build a `FileReader` in a practical scenario, we will have to add something in our documentation to point at a separate module entirely.
2. There’s no enforcement of the validity of `_fd` as a file descriptor; it’s just some integer, which the user could easily pass an incorrect instance of, with no error.

In isolation, `dataclass` by itself can’t solve all our problems, so let’s add in the second technique.

Using `classmethod` factories to create objects

We don’t want to require any additional imports, or require users to go looking at any other modules — or indeed anything other than `FileReader` itself — to figure out how to create a `FileReader` for its intended usage.

Luckily we have a tool that can easily address all of these concerns at once: `@classmethod`. Let’s define a `FileReader.open` class method:

```python
from typing import Self
@dataclass
class FileReader:
    _fd: int
    @classmethod
    def open(cls, path: str) -> Self:
        return cls(fileio.open(path))
```

Now, your callers can replace `FileReader("path")` with `FileReader.open("path")`, and get all the same benefits.

Additionally, if we needed to `await fileio.open(...)`, and thus we needed its signature to be `@classmethod async def open`, we are freed from the constraint of `__init__` as a special method. There is nothing that would prevent a `@classmethod` from being `async`, or indeed, from having any other modification to its return value, such as returning a `tuple` of related values rather than just the object being constructed.

Using `NewType` to address object validity

Next, let's address the slightly trickier issue of enforcing object validity.

Our type signature calls this thing an `int`, and indeed, that is unfortunately what the lower-level `fileio.open` gives us, and that's beyond our control. But for our *own* purposes, we can be more precise in our definitions, using NewType → https://docs.python.org/3.13/library/typing.html#newtype:

```python
from typing import NewType
FileDescriptor = NewType("FileDescriptor", int)
```

There are a few different ways to address the underlying library, but for the sake of brevity and to illustrate that this can be done with *zero* run-time overhead, let's just *insist* to Mypy that we have versions of `fileio.open`, `fileio.read`, and `fileio.write` which actually already take `FileDescriptor` integers rather than regular ones.

```python
from typing import Callable
_open: Callable[[str], FileDescriptor] = fileio.open  # type:ignore[assignment]
_read: Callable[[FileDescriptor, int], bytes] = fileio.read
_close: Callable[[FileDescriptor], None] = fileio.close
```

We do of course have to slightly adjust `FileReader`, too, but the changes are very small. Putting it all together, we get:

```python
from typing import Self
@dataclass
class FileReader:
    _fd: FileDescriptor
    @classmethod
    def open(cls, path: str) -> Self:
        return cls(_open(path))
    def read(self, length: int) -> bytes:
        return _read(self._fd, length)
    def close(self) -> None:
        _close(self._fd)
```

Note that the main technique here is not *necessarily* using `NewType` specifically, but rather aligning an instance's property of "has all attributes set" as closely as possible with an instance's property of "fully valid instance of its class"; `NewType` is just a handy tool to enforce any necessary constraints on the places where you need to use a primitive type like `int`, `str` or `bytes`.

In Summary - The New Best Practice

From now on, when you're defining a new Python class:

- Make it a dataclass[2].
- Use its default `__init__` method[3].
- Add `@classmethods` to provide your users convenient and discoverable ways to build your objects.
- Require that *all* dependencies be satisfied by attributes, so you always start with a valid object.
- Use `typing.NewType` to enforce any constraints on primitive data types (like `int` and `str`) which might have magical external attributes, like needing to come from a particular library, needing to be random, and so on.

If you define all your classes this way, you will get all the benefits of a custom `__init__` method:

- All consumers of your data structures will receive valid objects, because an object with all its attributes populated correctly is inherently valid.
- Users of your library will be presented with convenient ways to create your objects that do as much work as is necessary to make them easy to use, and they can discover these just by looking at the methods on your class itself.

Along with some nice new benefits:

- You will be future-proofed against new requirements for different ways that users may need to construct your object.
- If there are already multiple ways to instantiate your class, you can now give each of them a meaningful name; no need to have monstrosities like `def __init__(self, maybe_a_filename: int | str | None = None):`
- Your test suite can always construct an object by satisfying all its dependencies; no need to monkey-patch anything when you can always call the type and never do any I/O or generate any side effects.

Before dataclasses, it was always a bit weird that such a basic feature of the Python language — giving data to a data structure to make it valid — required overriding a method with 4 underscores in its name. `__init__` stuck out like a sore thumb. Other such methods like `__add__` or even `__repr__` were inherently customizing esoteric attributes of classes.

For many years now, that historical language wart has been resolved. `@dataclass`, `@classmethod`, and `NewType` give you everything you need to build classes which are convenient, idiomatic, flexible, testable, and robust.

Acknowledgments

Thank you to my patrons → https://blog.glyph.im/pages/patrons.html who are supporting my writing on this blog. If you like what you've read here and you'd like to read more of it, or you'd like to support my various open-source endeavors → https://github.com/glyph/, you can support my work as a sponsor → https://blog.glyph.im/pages/patrons.html! I am also available for consulting work → mailto:consulting@glyph.im if you think your organization could benefit from expertise on topics like "but what *is* a 'class', really?".

---

1. If you aren't already familiar, a "file descriptor" is an integer which has meaning only within your program; you tell the operating system to open a file, it says "I have opened file 7 for you", and then whenever you refer to "7" it is that file, until you `close(7)`. ↩
2. Or an attrs class → https://blog.glyph.im/2016/08/attrs.html, if you're nasty. ↩
3. Unless you have a really good reason to, of course. Backwards compatibility, or compatibility with another library, might be good reasons to do that. Or certain types of data-consistency validation which cannot be expressed within the type system. The most common example of these would be a class that requires consistency *between* two different fields, such as a "range" object where `start` must always be less than `end`. There are always exceptions to these types of rules. Still, it's pretty much *never* a good idea to do any I/O in `__init__`, and nearly all of the remaining stuff that may *sometimes* be a good idea in edge-cases can be achieved with a `__post_init__` → https://docs.python.org/3.13/library/dataclasses.html#dataclasses.__post_init__ rather than writing a literal `__init__`. ↩