asn目录（主要代码）

- asn
  - data　处理&管理数据
  - llm　编写prompt，提供llm调用接口
  - agent　实现agent模块和逻辑，提供功能接口
  - env　面向社交平台的模拟环境
- dataset　数据集
- config　配置文件
- log　运行日志
- simulator.py　模拟器demo

data（处理源数据）

1. 处理不同来源的数据为统一格式

```
class Data:
    """
        user attributes: id(str), info, posts, likes, following, followers
        post attributes: id(str), author_id, quote_id, timestamp, text, type
        meta attributes: other attributes such as history, etc.
    """
```

example.json

dataset > {} example.json > ...


```
{
    "users": [ ···
    ],
    "posts": [ ···
    ],
    "meta": {
        "history": { ···
        },
        "user_profile": { ···
        },
        "mastodon_info": { ···
        }
    }
}
```


```python
class DataTransformerBluesky:
    """
    Load and process data for initial simulation and testing. Coupling with data.
    Data format:
    name.json
    """
    def __init__(self, data_path):
        self.data_path = data_path
        with open(self.data_path, "r") as f:
            data = json.load(f)
        self.data = data


    def transform_data(self, num_users: int, strategy: str, time_window) -> Data:
        """
        Transform data to the format of the simulator
        time_window example: "2024-02-01 00:00:00=2024-02-28 23:59:59" 用于统计时间窗口内用户活跃度
        """

        users, posts, edges = self.data["users"], self.data["posts"], self.data["edges"]
        print("Users: %d, Posts: %d, Edges: %d" % (len(users), len(posts), len(edges)))
```

2. 筛选参与模拟的用户

```python
        elif strategy == "active":
            # 策略：选择活跃用户
            sampled_users = random.sample([user for user in users if user["class"] == "content creator"], num_users)
            if len(sampled_users) < num_users:
                sampled_users += random.sample([user for user in users if user["class"] == "active"], num_users - len(sampled_users))
            if len(sampled_users) < num_users:
                sampled_users += random.sample([user for user in users if user["class"] == "inactive"], num_users - len(sampled_users))
            users = sampled_users
            print("Content creator: %d, Active: %d, Inactive: %d" % (len([user for user in users if user["class"] == "content creator"]), len([user
        elif strategy == "active_by_time":
            # 根据时间区间内发帖数量判断用户活跃程度，并选择活跃用户
            time_begin, time_end = time_window.split("=")
            user2num_posts_by_time = {}
            for post in posts:
                # post["date"]: 202402292310   time_begin, time_end: "2024-02-01 00:00:00", "2024-02-28 23:59:59"
                post_time = datetime.strptime(str(post["date"]), "%Y%m%d%H%M")
                if time_begin <= datetime_to_str(post_time) <
                    user2num_posts_by_time[post["user_id"]] = user2num_posts_by_time.get(post["user_id"], 0) + 1
            # 增加随机扰动
            for key in user2num_posts_by_time:
                user2num_posts_by_time[key] += random.randint(-2, 2)
            # 根据发帖数排序
            user2num_posts_by_time = sorted(user2num_posts_by_time.items(), key=lambda x: x[1], reverse=True)
            # 选择前num_users个用户
            user2num_posts_by_time = user2num_posts_by_time[:num_users]
            users = [user for user in users if user["user_id"] in [user[0] for user in user2num_posts_by_time]]
            print("Content creator: %d, Active: %d, Inactive: %d" % (len([user for user in users if user["class"] == "content creator"]), len([user
        else:
            raise ValueError("Sampling strategy error: %s" % strategy)
```

(variable) user2num_posts_by_time: dict

TODO

构造llm-based数据集，构造一个类似DataTransformerVirt，修改Data类的属性或者把未定义属性放在metadata里

llm（llm工具）

1. llm invoke方法实现 & 模型管理

```python
class LLMManager:
    llm_name: str = "Local"
    llm_model: str = ""
    embed_name: str = "Local"
    embed_model: str = ""
    lora_path: str = ""
    llm: LLM = None
    embed_model: Embeddings = None

    @classmethod
    def set_manager(cls, conf) -> None:
        cls.llm_name = conf["llm_name"]
        cls.llm_model = conf["llm_model"] if "llm_model" in conf else ""
        cls.llm_url = conf["llm_url"] if "llm_url" in conf else "http://localhost:8001/v1"
        cls.embed_name = conf["embed_name"]
        cls.embed_model = conf["embed_model"] if "embed_model" in conf else ""
        cls.embed_url = conf["embed_url"] if "embed_url" in conf else "http://localhost:8002/v1"
        cls.lora_path = conf["lora_path"] if "lora_path" in conf else ""
        if cls.llm_name == "OpenAI":
            cls.llm = OpenAILLM(model=cls.llm_model, base_url=cls.llm_url, api_key=conf["api_key"] if "api_key" in 
        else:
            raise ValueError(f"Unknown model name: {cls.llm_name}")
        if cls.embed_name == "OpenAI":
            cls.embed_model = OpenAIEmbed(model=cls.embed_model, base_url=cls.embed_url, api_key=conf["api_key"] if
        else:
            raise ValueError(f"Unknown model name: {cls.embed_name}")
        print(f"LLM Manager set to: {cls.llm_name}, {cls.llm_model}, {cls.embed_name}, {cls.embed_model}")

    @classmethod
    def get_llm(cls) -> LLM:
        return cls.llm

    @classmethod
    def get_embed_model(cls) -> tuple[int, Embeddings]:
        return cls.embed_model.embed_size(), cls.embed_model
```

```python
class OpenAILLM(LLM):
    def _call(self, prompt: str, prompt_sys="You are a helpful assistant.", sft=False, **kwargs) -> str:
        self.task_ids.append(task_id)

        # 最多retry3次
        retry_count = 0
        while retry_count < 3:
            try:
                model = self.model if not sft else "sft"
                time_start_call = time.time()
                chat_response = self.client.chat.completions.create(
                    model=model,
                    messages=[
                        {"role": "system", "content": prompt_sys},
                        {"role": "user", "content": prompt},
                    ],
                    temperature=0.0,
                    timeout=3000
                )
                response = chat_response.choices[0].message.content
                time_end_call = time.time()
                get_logger().debug(f"PROMPT_SYS: {prompt_sys}\nPROMPT: {prompt}\n\nRESPONSE: {response} \n\nTask ID: {task_id}\
                self.task_ids.remove(task_id)
                # 保存 llm 的 prompt 和 response
                with lock_llm_save:
                    llm_out_json.append({
                        "prompt": prompt,
                        "response": response,
                    })
                    with open("llm_out.json", "w") as f:
                        json.dump(llm_out_json, f, indent=4)
                # 如果是think模型，筛掉think的内容
                if "<think>" in response and "</think>" in response:
                    response = response.split("</think>")[-1]
                return response
            except Exception as e:
                get_logger().error(f"Error in OpenAILLM: {e}\nTask ID: {task_id}\nPrompt: {prompt}")
                retry_count += 1
                time.sleep(3)
                continue

        # 失败则返回空字符串
```

2. prompt

```python
class Prompts:
    profile = \
"""
Please analyze the user's recent social media activities and interactions to identify their key characteristics and prefere

Consider the following aspects:
- What's the user's typical behavior on social media? Do they post frequently, like many posts, or share content often?
- What types of content do they engage with the most? Are there specific topics or themes that interest them?
- What values or beliefs do they express in their posts or interactions?

Requirements:
- Summarize the user's key characteristics and preferences based on their recent social media activities.
- If there is no activity record, that indicates the user is inactive and silent.
- Give a description of the user's characteristics and preferences in 1-5 sentences, capturing the main themes and behavior
- Response should be in 1-5 sentences within 200 words.

Your response should be in the second-person narrative like:
"You are a social media user who enjoys sharing your thoughts on technology and gaming. Your activity level is high, and yo

Here is the user's recent history of social media activities:
{history}
"""

    react_system = \
"""
    Act as a uer in social media platform. Your personal characteristics: "{characteristics}"
    Your should decide whether to "Like" or "Repost" a post pushed to your feed.
    "Like" means you like the post and want to show your appreciation. "Repost" means you want to share the post with more
    Consider the following aspects to decide:
- Does the post align with your interests and values?
```

agent（agent模块&逻辑实现）

- **plan**

  规划agent日常活跃时间

- **memory**

  管理agent记忆：write、retrieve

- **profile**

管理agent的profile：目前只有characteristic: str 一个属性，从真实用户的历史行为记录中初始化 （抽取agent在一段时间内的行为，整理为文本，由llm总结agent的characteristics）

TODO 如果是虚拟人物，可增加字段用于 存储自定义profile

- **action**

  定义行为空间

```python
class Act:
    type: str       # "read", "like", "retweet", "post"
    text: str
    timestamp: datetime
    def __init__(self, type: str, text: str, timestamp: datetime):
        self.type = type
        self.text = text
        self.timestamp = timestamp

    def __repr__(self):
        return f"Act(type={self.type}, text={self.text}, timestamp={self.timestamp})"

    def save_to_dict(self):
        return {
            "type": self.type,
            "text": self.text,
            "timestamp": datetime.strftime(self.timestamp, "%Y-%m-%d %H:%M:%S"),
        }
```

  提供 write_post, react_to_post, react_to_posts 等接口

- **agent**

  封装Agent逻辑，提供功能接口：make_plan, replay, generate, recieve

```python
class LLMAgent(Agent):
    def __init__(self, info: Optional[dict] = None, memory = None, profile = None):
        super().__init__()
        self.info = info
        self.llm = LLMManager.get_llm()
        # modules
        self.action = ActionModule()
        if memory is None:
            memory = NaiveMemoryModule()
        self.memory = memory
        self.profile = ProfileModule() if profile is None else profile
        self.plan = Plan()
        self.behavior_record = []
```

```
61
62  >   def add_to_memory(self, observation, now):…
66
67  >   def recieve(self, text: str, now: datetime, update_memory=True, incontext=False, fake_history=None, log=None):…
96
97  >   def recieve_all(self, texts, now: datetime, update_memory=True, incontext=False, fake_history=None, log=None):…
134
135 >   def generate(self, now: datetime, previous_posts, update_memory=True, force=False, incontext=False, fake_history=None, log=None):…
153
154 >   def make_plan(self, now: datetime):…
159
160 >   def replay(self, act):  # TODO…
178
179 >   def replay_batch(self, acts, timestamp: datetime):…
```

env（社交平台系统模拟）

- **environment（定义用户、消息、环境接口）**

  模拟社交媒体环境，管理消息池，管理用户

```python
class User:
    def __init__(self, id, info, agent: Agent, mastodon_info: dict, following=[], followers=[]):
        self.id = id
        self.info = info
        self.agent = agent
        self.following = following
        self.followers = followers
        self.posts = []
        self.likes = []
        self.reposts = []
        self.mastodon_info = mastodon_info

    def get_following_ids(self):…

    def get_follower_ids(self):…

    def get_status_ids(self):…

    def get_like_ids(self):…

    def get_repost_ids(self):…

    def post(self, status, created_at: Optional[datetime] = None):…

    def like(self, status_id):…

    def repost(self, status_id):…
```

```python
class Message:
    def __init__(self, id, type, text, author_id, timestamp, quote_id=None, mastodon_info=None):
        self.id = id
        self.type = type # "post" or "repost"
        self.text = text
        self.author_id = author_id
        self.timestamp = timestamp
        self.quote_id = quote_id
        self.mastodon_info = mastodon_info
        self.embed = None
        self.liked_by = []
        self.reposted_by = []

    def origin_id(self):
        # origin_id 记录 转发或引用的原始消息的 id，mastodon逻辑：转发消息时转发的其实是最初的原始消息
        return self.quote_id if self.quote_id else self.id
```

- **recommender**

  针对指定用户，对内容列表进行排序，依据热度、匹配度（相似度）、时间

Simulator.py

封装了模拟demo所需的基础功能

```
58      # Initialize enviroment from data
59  >   def init_env_from_data(self, env: Environment, data: Data):…
72
73
74      # 根据历史记录，重演agent行为，初始化agent的memory模块
75  >   def replay_history(self, env: Environment, data: Data, time_begin: str, time_end: str, interval: str, parallel=True):…
152
153
154     # 初始化每个用户的profile
155 >   def get_users_profile(self, env: Environment, data: Data, time_begin: str, time_end: str, interval: str, parallel=True):…
177
178
179 >   def simulate_user(self, user: User, env: Environment, data: Data, conf: dict):…
228
229
230 >   def simulate_step(self, time_step):…
246
247
248 >   def simulate(self):…
```

```
python simulator.py
```

```
312     # Simulation
313     simulator = Simulator(conf)
314     simulator.simulate()
```

show.ipynb

将模拟结果推送到mastodon服务器做展示

Robert_Harrison
@Robert_Harrison

2024年2月11日

Oh FFS 空袭警报你去Fuck你自己吧，连同那些俄国人。 屁眼儿卖屌的俄国。

#UkrainianView

0

Kelly_Barton_DVM
@Kelly_Barton_DVM

2024年2月11日

首先你说你如何向房屋发射火箭，然后你去接受惩罚。因果报应
#UkrainianView

0

Amber_Rios
@Amber_Rios

2024年2月11日

如果按照普京的逻辑，我们应该把南美还给非洲。
#UkrainianView

0

requirements

Process

1. **处理数据**为指定格式
2. 随机**采样用户**（TODO1 随机采样=>采样活跃用户）
3. **模拟**
    1. 根据用户历史数据，**初始化**Agent的profile、memory模块。
    2. **轮式模拟**（TODO2 轮式模拟=>agent主动活跃。轮式模拟不合理，相当于手动设定了所有agent的统一活动频率）
        1. 为每个agent**分发内容**（msg，其他agent产生的帖子），利用关注关系和推荐系统
           （TODO3 分发数量暂时由config文件超参手动指定，热度和相似度、时间因素权重怎么设计）

2. agent对读取的内容**做出反应**（like、repost）

   （TODO4 反应行为空间不含comment，一是因为历史数据中没有comment；二是因为comment的加入会增加帖子的复杂度，加入comment后以什么形式展示给agent读者（全部/折叠），是否允许迭代的评论等问题）

3. agent**产生新内容**（post）agent判断是否要生成内容以及生成什么内容

   （TODO5 缺乏外界信息注入，依据记忆生成的内容依赖于其他agent产生的内容。真实情境下，发帖常常来源于现实世界中的事、新闻等。所以模拟相比于现实，生成的内容没有新鲜事，可能趋向于空洞/一致
   比如event（假设发生在agent、multiagent身上）注入）

TODO6 memory机制设计&实现（以帖子内容为key的解决任务式记忆/长短期记忆/others）

TODO7 实验设计

TODO8 手动投放消息接口

TODO9 展示消息投放后：传播路径、opinion变化。

TODO10 文档