Theme #1 - Multi-Agent Interactions

Environments for this theme involve cooperation, competition, negotiation, and coalition formation. Learning from these environments will enable agents to model the beliefs and incentives of others in partially observable settings. This drives theory-of-mind reasoning and emergent strategic behavior.

Expected Outcome: an environment that can be used to train multi-agent task handling in a LLM

Example environments: Market simulations, compute-allocation negotiations, collaborative puzzle worlds, mixed cooperative/competitive strategy games.

Theme #2 - (Super) Long-Horizon Planning & Instruction Following

You will build environments that require deep, multi-step reasoning with sparse or delayed rewards. After using these environments, the goal is to enable agents to decompose goals, track state over extended trajectories, and recover from early mistakes. The aim is to push beyond shallow next-token reasoning toward structured planning and durable internal representations.

Expected Outcome: an environment that can capture and improve LLM behaviour on challenging long horizon tasks that need long running sessions beyond context memory limits.

Example environments: (Think of OpenClaw workflows with Multi-turn tasks). Research-planning simulators, large-scale codebase refactoring tasks, strategic resource management worlds, long-horizon logistics optimization, extremely complicated long-horizon instruction following (e.g., 300 instructions scattered around).

Theme #3 - World Modeling

#3.1 Professional Tasks

Here you will develop environments that require real interaction with tools, APIs, or dynamic systems where the model is expected to do real hard work instead of exploiting short-cuts to arrive at the desired outcome. Learning from these environments will enable agents to maintain consistent internal state, update beliefs based on outcomes, and orchestrate multi-step workflows. The goal is to strengthen causal reasoning and persistent world models.

Expected Outcome: an environment capturing nuances of a defined partially observable world and improve LLM interaction with it

Example environments: Dynamic browser/API ecosystems, enterprise applications, scientific workflow loops (papers → code → experiments), economic simulations with feedback, tool-discovery benchmarks.

#3.2 Personalized Tasks

Here we will develop an environment that offers real personalized task handling, imagine replying to personal messages or handling dinner conflicts due to work conflicts, replying to tough emails. Think any personal assistant tasks

Expected Outcome: An environment that gives the model a realistic simulation of handling personal tasks, conflicts and managing them as delegations

Example environments: Executive Assistant Meeting Planner, Dinner and drive planning, email and message replying, shopping, etc

Theme #4 - Self-Improvement

The focus here is to create environments where agents can learn to generate new challenges, escalate difficulty, and improve through self-play or adaptive curricula. Rather than optimizing fixed tasks, the goal is for agents to learn to drive their own capability growth. The objective is recursive skill amplification.

Expected Outcome: an environment for improving self-play of a LLM over a defined set of tasks

Example environments: Self-play negotiation arenas, auto-generated math/proof tasks, evolving coding competitions, adaptive RL curricula.

Theme #5: Wild Card - Impress Us!

We do not want to limit your focus if your idea doesn't fit the boxes above, we want and WILL reward out of box tasks, please be creative but remember to add submissions that meaningfully add value to LLM training on a certain task.

Guidelines for Problem Statement

- It is **NOT** mandatory to choose the same problem statement as Round 1. Only choose the same problem statement if it aligns with the above provided Hackathon themes.
- You can start working on your problem statement once you have finalized it. Post-training can be done onsite on 25th & 26th when you receive compute credits for HuggingFace.
- Before the onsite, we suggest you work on building the environment, agent behaviours, reward model and evaluate if your work aligns with the judging criteria given below.

Judging Criteria

Minimum requirements:

- Usage of OpenEnv (latest release)

- Show a minimal training script for your environment using Unsloth or HF TRL in Colab
- Write a mini-blog on HuggingFace or mini-video on YouTube talking about your submission, <2 minutes
- Your OpenEnv compliant environment should be hosted on Hugging Face Spaces.

Judging Overview

- **Evaluation:** Teams will be scored based on the following criteria:
    1. **Environment Innovation (40%):** Is the environment novel, creative, or challenging? Does it meaningfully test the agent's behavior?
    2. **Storytelling (30%):** Does the team clearly explain the problem, environment, and agent behavior? Is the demo engaging and easy to follow?
    3. **Showing Improvement in Rewards (20%):** Does the demo provide observable evidence of training progress (reward curves, metrics, or before/after behavior)?
    4. **Reward and Training Script/Pipeline Setup (10%):** Is the reward logic coherent, and does the pipeline produce meaningful improvement in the agent's inference (how it acts in the environment)?

OpenEnv Hackathon - What Judges Look For

This guide tells you what makes a strong submission for the OpenEnv Hackathon (India 2026).
Read it before you start building, and again before you submit.

For the list of themes and example problems, refer to the top sections.

NOTE: Please remember only one submission per team. If you have multiple ideas, pick the best one and go for it. Please make sure that the URL link of your environment is

submitted as judges will pull the environment from the URL to evaluate it. Changes or commits after the submission deadline will not be considered.

TL;DR

Build an environment that an LLM could actually be trained on to get measurably better at
something interesting. Then show that training. Then tell the story.

A messy but ambitious environment with real training evidence beats a polished but boring one.
Pick a problem that excites you (that energy comes through in the pitch).

Judging Criteria

Criterion: Environment Innovation
Weight: 40%
What it means:
Is the environment novel, creative, or genuinely challenging?
Does it meaningfully test agent behavior **in** a way that hasn't been done before?

Criterion: Storytelling & Presentation
Weight: 30%
What it means:
Can you clearly explain the problem, the environment, and what the agent learned?
Is the demo engaging and easy to follow **for** a non-technical audience?

Criterion: Showing Improvement in Rewards
Weight: 20%
What it means:
Is there observable evidence of training progress? Reward curves, before/after behavior,

comparison against a baseline -- anything that proves the agent learned something.

Criterion: Reward & Training Pipeline
Weight: 10%
What it means:
Is the reward logic coherent? Does the pipeline produce meaningful improvement **in** the trained
agent's behavior?

Minimum Submission Requirements

NOTE: These are **non-negotiable**. Submissions missing any of these are at a serious disadvantage.

- ☐ **Use OpenEnv** (latest release). Build on top of the framework; don't reinvent the wheel.
- ☐ **A working training script** using **Unsloth or Hugging Face TRL**, ideally as a Colab notebook so judges can re-run it.
- ☐ **Evidence that you actually trained**; at minimum, loss and reward plots from a real run.
- ☐ **A short writeup**: a mini-blog on Hugging Face or a < 2 minute video on YouTube explaining what your environment does and what you trained, or a short slide deck of presentation. Please make sure that all materials are linked from your README file so that judges can access them easily.
- ☐ **Push your environment to a Hugging Face Space** so it's discoverable and runnable.
- ☐ **A README** that motivates the problem, explains how the env works, and shows results.
    - ☐ README should have a link to the environment in the Hugging Face Space. It should also have all additional references to other materials (e.g. videos, blog posts, slides, presentations, etc.) that you want to include.
- ☐ Please do not include big video files in your Env submission on HF Hub as we would like to have a small size for each env (Please use url as reference link to additional materials).

What Makes a Submission Stand Out

Pick an ambitious, original problem

The themes (problems) are deliberately open. Use them as launching pads, not boxes. Judges have seen a lot of chess, snake, tic-tac-toe, and grid-world clones. To score well on innovation,
you need a genuinely fresh angle. Some questions to ask yourself:

- Does this environment exist to teach an LLM something it currently can't do well?
- Is the domain underexplored in RL/LLM training?
- Could a researcher write a paper about training on this?

Design a reward signal that actually teaches

A great environment has a reward function that:

- Provides a **rich, informative signal** (not just 0/1 at the end)
- Captures something **hard to measure** in a clever way
- Uses OpenEnv's **Rubric system** thoughtfully (composable rubrics > monolithic scoring)
- Is **hard to game**; an agent that exploits the reward without solving the task should not get high scores

Show real training, end to end

The bar isn't "training script exists." The bar is "training script runs against the environment, the
agent learns, and you can show it." Concretely:

- Your training loop should connect to **your** environment (not a static dataset)
- Train long enough that the curves mean something
- Compare a **trained agent vs. a random/untrained baseline**; quantitative and/or qualitative
- Include the plots and numbers in your README and writeup

Make your plots readable

Reviewers spend seconds, not minutes, on each plot. Help them out:

- **Label both axes** (e.g. “training step” / “episode” on x, “reward” / “loss” on y) and include units where they apply
- Save plots as *.png* or *.jpg* and **commit them to the repo** (don’t leave them only in a Colab cell or a deleted Wandb run) (if you ran via Wandb, please include the link to that specific run of your plots)
- **Embed the key plots in your README** with a one-line caption explaining what each one shows If you have multiple runs (baseline vs. trained, ablations, etc.), put them on the same axes so the comparison is obvious

Tell a story, not an API doc

Your README, blog, and pitch should answer:

1. **Problem)** what capability gap or interesting domain are you targeting?
2. **Environment)** what does the agent see, do, and get rewarded for?
3. **Results)** what changed after training? Show it.
4. **Why does it matter)** who would care, and why?

A reviewer should be able to read your README in 3~5 minutes and want to try your environment.

NOTE: If you have a video, HF post, or anything else interesting, please make sure that it’s linked
from your README as a link.

Engineer it cleanly (table stakes)

Engineering quality matters less than ambition, but sloppy work hurts. Make sure you:

- Use OpenEnv’s Environment / MCPEnvironment base classes properly
- Respect the **client / server separation** (clients should never import server internals)
- Follow the standard Gym-style API (reset, step, state)
- Have a valid openenv.yaml manifest
- Don’t use reserved tool names (reset, step, state, close) for MCP tools

Final Note

Judges are looking for environments that push the frontier of what we can train LLMs to do. Be
ambitious. Pick a problem you find genuinely interesting; that almost always produces better
work than chasing what you think judges want. Good luck.