👾 Overview

When creating or solving engineering challenges:

- Select or create a **real** GitHub issue
- Ensure the task is **self-contained**
- Provide or adhere to **objective tests**
- Maintain **deterministic**, **verifiable** behavior
- Follow **repository patterns and architecture**

> The goal: Test skills on real-world workflow — no ambiguity, no guesswork.

🏗️ Repository Requirements

A repository is **eligible** if:

- ⭐ 500+ GitHub stars
- 🌍 Public repo (readable & forkable)
- 🐍 Main language: Python or TS
- 🏭 Used by real developers / maintained recently
- 🧩 License is permissive/compatible
- 📌 Stable commit hash provided (no moving target)

🎯 Difficulty & Scope Alignment

Avoid tasks that are:

🚫 **Too small**
• Typos
• Renames
• Single-line bug fix

🚫 **Too large**
• Full system redesign
• Repo-wide migrations

✔ **Target Scope**

- Clear boundaries
- Represents real engineering work
- No more than 1–2 modules touched where possible

✔ Complexity Targets

Difficulty	Expected Time
Medium ⚡	1.5–4 hrs
Hard 🔥	4+ hrs

The goal is: **Human-achievable, AI-challenging** (≤20% AI auto-solve)

✍️ Writing a Strong Problem Description

Focus on **WHAT** and **WHY**

Do:

- Describe intended functionality
- Explain expected user/system behavior
- Provide examples if helpful
- Declare expected **outputs**, **errors**, **edge cases**

Avoid:

- Procedural “how-to” instructions
- Naming conventions unless required
- Test-file references (“check X in Y test”)

⚙️ Determinism Requirements

Every requirement must be:

- **Objectively testable**
- **Fully defined**
- **Independent of randomness**

❌ Failing Examples:

- Time-based checks without mocking
- External remote API calls
- Tests depending on internal state order

✔ Prefer:

- Response structure
- Exceptions
- Return values
- Documented interface behavior

🧩 Full Problem Evaluation Rubric

The following criteria determine **whether a problem is accepted**:

Category	Fails ❌	Passes 👍
Problem Quality	Missing context, unclear, poorly formatted	Clear, professional, self-contained and readable
Determinism	Vague, untestable behavior, mismatch with tests	Fully objective + aligned with tests
Scope	Too small or too large	Real engineering effort, well-bounded
Difficulty	Trivial or impossible	Medium/Hard — solvable with thought
Test Quality	No alignment, flaky, random behavior	Full coverage, deterministic, baseline pass/new fail

A problem must score **PASS in all 5** to be accepted.

📌 Writing: Required Problem Structure

```
Problem Brief:
Explain the functionality, purpose, and success outcome.
Keep the prompt short, actionable, and unambiguous. Avoid
implementation steps — describe what must happen, not how to do it.
```

❌ DO NOT include:

- Test code references
- Shell commands
- Pseudocode
- Implementation hints

🧪 Test Patch Requirements

Your test submission **must include**:

✔ `test.sh` runner with 2 modes

```
./test.sh base   → run existing tests (should pass)
./test.sh new    → run new tests only (should fail initially)
```

✔ New tests must:

- Fail on original commit
- Test only specified behavior
- Use **deterministic** assertions
- Have complete edge-case coverage

✔ Tests must **not rely on**:

- Timers without mocking
- Network calls outside repo boundaries
- Random values w/o seeded determinism

✅ Example test.sh (TypeScript version)

```
#!/bin/bash
set -e

case "$1" in
  base)
    npm test -- tests/existing.test.ts
    ;;
  new)
    npm test -- tests/new.test.ts
    ;;
  *)
    echo "Usage: ./test.sh {base|new}"
    exit 1
    ;;
esac
```

If any **base** **tests fail**, remove them from the runner.
Known failing tests must not be included.

🛠️ Dockerfile Requirements

Your Dockerfile must:

✔ Allowed	❌ Forbidden
Copy repo	Installing package managers
Install deps	Automatically running tests
Base dev tools image	Using any other container base

✔ Valid Example:

```
FROM ..
WORKDIR /app
COPY . .
RUN npm install
CMD ["/bin/bash"]
```

> This Dockerfile must serve only as a **dev environment**, nothing more.

🧑‍💻 Final Problem Creator Checklist

Before submitting:

- Repo meets star + language + license rules
- Problem description ≤ **300 words** and self-contained
- Behavior fully defined and objectively testable
- Baseline tests: **PASS**
- New tests: **FAIL**
- Dockerfile meets rules
- Difficulty classification appropriate
- No hidden requirements outside description

🚀 Solution Contributor Requirements

To submit a valid solution:

✔ Implement **every** requirement stated in the description
✔ No contradictions or missing behaviors
✔ Follow **repository architecture + coding standards**
✔ Keep diff **clean** — no unrelated edits
✔ All tests (baseline + new) must pass
✔ Submit **patch file only**

> If code fixes unrelated bugs or touches extra files → **Rejected**

🔍 Full Solution Evaluation Rubric

Category	What Reviewer Looks For
Comprehensiveness	Did you implement **ALL** stated requirements with no omissions?
Code Quality	Clean, readable, correct patterns, no smells, no dead code
Regression Safety	No breaking existing features; all old tests must pass

---

❌ Automatic Rejection Examples

- Commented-out code left behind
- Adding unnecessary dependencies
- “Quick hack” instead of following existing conventions
- Changing unrelated functionality in passing
- Solution succeeds by **skipping** or **removing** tests

🧩 Patch Format Requirements

Your solution must be submitted as a `.patch` file created using:

```
git add .
git diff --cached > solution.patch
```

Patch must contain ONLY solution changes
(no test or Dockerfile modifications in solution submission)

🤖 Automated Validation Checks (Must Pass All)

Automated Check	What it Verifies
Baseline Test Execution	Base commit is healthy + reproducible
Test Patch Format Check	test.sh + patch diff are valid
Problem & Test Quality	All rules / rubric alignment
AI Difficulty Scoring	AI attempts must fail ≥ 80%
Precision & Alignment	All tests reflect description and vice-versa
Solution Format Check	Proper `.patch` formatting
Solution Test Execution	New tests pass only **after** solution

If ANY of these fail → requires revisions

👩‍💻 Final Solution Checklist

Before submitting:

- All stated behaviors implemented exactly
- All tests pass locally
- Only solution code changed (tests untouched)
- Code matches repo patterns and style
- No debugging output (`console.log`, prints)
- No unnecessary refactoring or files changed
- Patch clean and minimal

🚫 Common Pitfalls — Avoid These

Mistake	Result
Fixing failing tests instead of fixing code	❌ Rejected
Hidden behavior implemented but not described	❌ Ambiguity + Fail
Skip tests to pass CI	❌ Auto Rejected
Adding todo comments everywhere	❌ Code quality fail
Modifying repo tooling or CI config	❌ Hard reject

Deliverables: A zip file of your problem set!

1. Problem.md (should contain Title, description, language, category, difficulty, github repo link, issue url and commit hash)
2. Test.patch
3. Solution.patch
4. Dockerfile

🎯 The Golden Rule

> **Everything tested must be described.**
> **Everything described must be tested.**

No more. No less.

Permissible liscences-

MIT License (MIT) Apache License 1.0 (Apache-1.0)
Apache License 1.1 (Apache-1.1)
Apache License 2.0 (Apache-2.0)
Apache-2.0-Modified
Apache-with-LLVM-Exception
Apache-with-Runtime-Exception BSD-1-Clause
BSD 2-Clause "Simplified" License (BSD-2-Clause)
BSD-2-Clause-Flex Any external distributions must include the or clause specified in the license. BSD 2-Clause FreeBSD License (BSD-2-Clause-FreeBSD)
BSD-2-Clause-Modification
BSD-2-Clause Plus Patent License (BSD-2-Clause-Patent)
BSD 2-Clause with views sentence (BSD-2-Clause-Views)
BSD 3-Clause "New" or "Revised" License (BSD-3-Clause)
BSD with attribution (BSD-3-Clause-Attribution) BLAS BSD-3-Clause-EricHeitz
BSD-3-Clause-HealthLevelSeven
Lawrence Berkeley National Labs BSD variant license (BSD-3-Clause-LBNL)
BSD-3-Clause-Modification
BSD-3-Clause-OpenMPI
BSD-3-Clause-plus-CMU-Attribution
BSD-3-Clause-plus-Paul-Mackerras-Attribution
BSD-3-Clause-plus-Tommi-Komulainen-Attribution
BSD 4-Clause "Original" or "Old" License (BSD-4-Clause)
BSD-4-Clause-Argonne
BSD-4-Clause-Atmel
BSD-4-Clause-Giffin
BSD 4-Clause PC/SC Lite for Suse (BSD-4-Clause-PC-SC-Lite)
BSD-4-Clause-Plus-Modification-Notice
BSD-4-Clause (University of California-Specific) (BSD-4-Clause-UC)
BSD-4-Clause-Visigoth
BSD-4-Clause-Vocal
BSD-4-Clause (Wasabi-Specific) (BSD-4-Clause-Wasabi)
BSD 4.3 TAHOE License (BSD-4.3TAHOE)
BSD-5-Clause BSD-FatFs
BSD-Mixed-2-Clause-And-3-Clause BSD Protection License (BSD-Protection) BSD Source Code Attribution (BSD-Source-Code)
Boost Software License 1.0 (BSL-1.0) Creative Commons Attribution 1.0 Generic (CC-BY-1.0)
Creative Commons Attribution 2.0 Generic (CC-BY-2.0)
Creative Commons Attribution 2.5 Generic (CC-BY-2.5)
Creative Commons Attribution 3.0 Unported (CC-BY-3.0)

Creative Commons Attribution 4.0 International (CC-BY-4.0)
GNU-All-permissive-Copying-License GNU General Public License v2.0 w/Autoconf exception (GPL-2.0-with-autoconf-exception) GNU General Public License v2.0 w/Classpath exception (GPL-2.0-with-classpath-exception) GNU General Public License v3.0 w/Autoconf exception (GPL-3.0-with-autoconf-exception)