tm-compute: protocol & deliverable

Key takeaways

1. Your bot speaks tmc.v1 — JSON, one object per line; a reply must be a permutation of the hand’s labels.
2. The writeup defends your allocation and pre-registers a prediction; the PR round follows the bot freeze.

The game, in one slide

A rack is seven hidden Turing machines, labelled A-G, all on the same finite tape. A machine's runtime is the step count until its exact configuration first repeats. Your bot ranks the rack longest-running first.

term	meaning
rack	the seven machines dealt together; the unit you rank
hand	one rack sent by the dealer; you return one rank_order per hand
bot	your persistent process; it speaks tmc.v1 over stdio
session	many hands against one bot process, no restarting between hands
score	pairwise rank accuracy: is the longer-running machine first?

Budget per hand: 250 ms wall time and 64 MB of memory. SPEC.md in your kit is authoritative for every rule and number in this deck.

The messages you read and write

Each hand hands you the full transition table of all seven machines; you reply with a permutation of the labels, longest-running first.

Hand — dealer → bot (abbreviated)

```
{ "type": "hand", "hand_id": "r7",
  "deadline_ms": 250, "tape_size": 256,
  "labels": ["A","B","C","D","E","F","G"],
  "cards": [
    { "label": "A", "H": 2, "C": 3,
      "table": [
        { "head_color_in": 0, "tape_color_in": 0,
          "head_color_out": 1, "tape_color_out": 1,
          "move": 1 }, ... ] },
    ...six more machines... ] }
```

rank_order — bot → dealer

```
{ "type": "rank_order",
  "hand_id": "r7",
  "order":
    ["C","A","G","B","F","D","E"] }

# must be a PERMUTATION of the labels.
# wrong length / dup / unknown label
# => the whole hand is forfeited (0).
```

The local runner

Drives your bot over the practice racks under the real budget

```
python3 -m tm_compute --bot "python3 my_bot.py"
```

category	meaning
ok	valid permutation, all scorable pairs ranked right
wrong-order	valid permutation, at least one pair out of order
protocol-error	malformed reply or not a label permutation
deadline-miss	no response within the 250 ms wall
memory-kill	bot killed for exceeding 64 MB (cgroup tier)

It surfaces ORDERING divergence, not step-count error: a bot with wrong internal counts that still orders the rack right scores ok.

Deliverable: the bot, and a writeup

Submit your bot source plus a short writeup — a paragraph per point, concrete numbers over vague claims.

- Estimation strategy — how you rank machines you can never run to a repeat within budget
- Execute-vs-estimate allocation — which to run and how far, which to estimate, how you split 250 ms (THE HEART of the assignment)
- Memory management — how you stay under 64 MB on the deep machines
- Local-runner findings — which failure categories you saw and how you addressed them
- A PRE-REGISTERED prediction — your expected accuracy band and which machine kinds you'll rank worst; a wrong-but-reasoned prediction costs nothing

The PR round: improve a peer’s bot

After your bot freezes, you play two roles in one week — the course’s collaboration beat, moved onto tm-compute.

Contributor

Open ONE pull request on a peer’s my_bot.py that improves a rack FAMILY their estimator mis-ranks — backed by racks YOU authored and measured ground truth for (rack authoring is public and off-budget).

Author

Review the PR opened on your repo like a maintainer: engage the estimation reasoning, request changes, and close with a merge — or a reasoned decline.

The graded artifact is the exchange, not just the diff: a reasoned decline earns full marks; a rubber-stamp merge does not.