1RM Predictions All 8 Prompts

Treat them like it's a recipe.

Cursor Prompt — Chunk 1: Minimal 1RM Calculator (with tests & git)

You are Cursor. Create a minimal, well-tested TypeScript CLI app that computes estimated 1RM using the Epley formula from the provided spec: `est1RM = weight * (1 + reps/30)`. This is **Chunk 1** of the project — the simplest version that takes basic inputs and outputs a value. Do **not** implement later chunks yet.

Requirements

- **Language/Stack:** Node.js + TypeScript.
- **Interface:** Simple CLI (`npx 1rm <weight> <reps> [--json]`).
- **Formula:** If `reps === 1`, return `weight` as the true 1RM; otherwise compute `Math.round(weight * (1 + reps/30))` to nearest lb.
- **Validation:**
  - `weight` must be a positive number.
  - `reps` must be an integer `>= 1` and `<= 30`.
  - On invalid input, print a helpful error and exit with non-zero code.
- **Output:**
  - Default: human-readable line like `Estimated 1RM: 263 lb`.
  - If `-json`: `{ "weight": 225, "reps": 5, "estimated1RM": 263, "method": "epley" }`.
- **Rounding:** Round to the nearest pound (use `Math.round`).
- **Tests:** Unit tests covering happy paths and validation errors.
- **Git protocol:** Conventional Commits, branch per chunk, clean history.

Project Scaffolding

Create this file/folder structure:

```
1rm-calculator/
  ├─ src/
  │   ├─ index.ts          # CLI entrypoint (parses args, prints results)
  │   └─ calc.ts           # pure function(s) for 1RM calculations
  ├─ test/
  │   └─ calc.test.ts      # unit tests for calc.ts
  ├─ package.json
  ├─ tsconfig.json
  ├─ .gitignore
  └─ README.md
```

Implementation Details

- `calc.ts` exports:
  - `export function estimate1RM(weight: number, reps: number): number` implementing Epley and rounding to nearest integer.
  - Throw `Error` on invalid inputs (negative/zero weight, reps < 1, non-integer reps, reps > 30).
- `index.ts`:
  - Use a tiny argument parser (no dependency) or Node `process.argv`.
  - Support `-json` flag.
  - Catch validation errors, print message, exit code `1`.
- `package.json` scripts:
  - `build`: `tsc`
  - `start`: `node dist/index.js`
  - `test`: `vitest run` (use Vitest) or `jest` – choose Vitest.
  - `dev`: `vitest`

    - `lint`: `eslint .` (optional for this chunk)
- **Dependencies:** `typescript`, `vitest`, `@types/node` (dev). No runtime deps required.
- Publish an `npm` bin via `package.json` for local use: set `"bin": { "1rm": "dist/index.js" }` and make the built file shebang `#!/usr/bin/env node`.

Tests (Vitest)

Write comprehensive tests in `test/calc.test.ts`:

1. **Epley example**: `estimate1RM(225, 5)` → `263` (since `225 * (1 + 5/30) = 262.5`, rounded to `263`).
2. **True single**: `estimate1RM(265, 1)` → `265`.
3. **Minimum valid reps**: `estimate1RM(135, 1)` → `135`.
4. **Upper reps bound**: `estimate1RM(135, 30)` should be `Math.round(135 * (1 + 1)) = 270`.
5. **Invalid weight**: `estimate1RM(0, 5)` throws.
6. **Invalid reps (0)**: throws.
7. **Invalid reps (non-integer)**: e.g. `estimate1RM(200, 3.5)` throws.
8. **Invalid reps (>30)**: throws.

Also add an integration test (optional in this chunk) that spawns the CLI with `225 5 --json` and asserts on stdout JSON.

README.md

Include:

- Short overview of the project and this chunk.
- Usage examples:
    - `npx 1rm 225 5` → `Estimated 1RM: 263 lb`
    - `npx 1rm 265 1 --json` → `{ ... }`
- Notes on formula and rounding.
- How to run tests.

Git Protocol

1. Init repo and branch for this chunk:

```
git init
git checkout -b feat/chunk-1-minimal-1rm
```

2. Make atomic commits using Conventional Commits:
   - `feat(calc): add estimate1RM with epley formula`
   - `feat(cli): implement CLI with --json flag`
   - `test(calc): cover edge cases and examples`
   - `docs(readme): usage examples`
3. Push branch and open PR to `main` titled: **feat: Chunk 1 – Minimal 1RM calculator** with a checklist:
   - ☑ ~~Pure function implemented~~
   - ☑ ~~CLI implemented~~
   - ☑ ~~Tests passing (Vitest)~~
   - ☑ ~~README explains usage~~
4. After tests pass and review (self-review for now), merge with a squash commit message: `feat: chunk 1 minimal 1RM calculator`.
5. Tag the merge commit: `git tag -a v0.1.0 -m "Chunk 1 minimal 1RM" && git push --tags`.

Commands to Run

```
# from project root
npm init -y
npm i -D typescript vitest @types/node
npx tsc --init --rootDir src --outDir dist --esModuleInterop --module commonj
s --target ES2020
mkdir -p src test
# create src/calc.ts, src/index.ts, test/calc.test.ts, README.md per above
```

```
npm run build
npm test
# local CLI try
node dist/index.js 225 5
node dist/index.js 225 5 --json
```

Acceptance Criteria for Chunk 1

- All unit tests pass locally with `npm test`.
- CLI prints the expected values for the example inputs.
- Invalid inputs produce clear errors and non-zero exit codes.
- Git history reflects clean, conventional commits on a feature branch, merged into main, tagged `v0.1.0`.

Stop after this. Do not implement data collection, weekly grouping, or modeling yet. When the user confirms Chunk 1 works, we will proceed to Chunk 2.

Cursor Prompt — Chunk 2: Raw Data Collection (with tests & git)

You are Cursor. Extend the existing **TypeScript CLI 1RM calculator** from Chunk 1 by adding **raw data collection** capabilities. This chunk focuses on capturing **all essential training variables** needed to build a structured dataset for future model training. This is **Chunk 2** — do **not** implement later chunks yet.

Requirements

- **Language/Stack:** Node.js + TypeScript (continue from Chunk 1).
- **Goal:** Capture and store structured workout data including all key raw inputs.
- **New Inputs (6 total):**
  1. `reps`
  2. `weight`

3. `sets`
4. `exerciseName` → e.g., `"bench_press"`, `"incline_smith"`, `"lat_pulldown"`
5. `exerciseType` → e.g., `"barbell"`, `"cable"`, `"machine"`
6. `date` → default = current date unless user provides `-date YYYY-MM-DD`

- **CLI Flags:**
  - `-sets <n>`
  - `-exercise <string>`
  - `-equipment <string>`
  - `-date <ISO or YYYY-MM-DD>`
  - `-save` → Save full session entry
  - `-list [n]` → List past n sessions
  - `-json` → JSON output for both save result and list
- **Session schema:**

```
{
  "date": "2025-11-10T00:00:00.000Z",
  "exerciseName": "bench_press",
  "exerciseType": "barbell",
  "sets": 3,
  "weight": 225,
  "reps": 5,
  "estimated1RM": 263,
  "method": "epley"
}
```

- **Validation:**
  - `sets` must be integer ≥ 1
  - `exerciseName` required
  - `exerciseType` required

- Missing date → auto-generate ISO timestamp
- Invalid date → error
- Weight/Reps still validated using Chunk 1 rules

Project Scaffolding

Add or update files:

```
1rm-calculator/
  ├─ src/
  │   ├─ index.ts          # update CLI to handle new flags + saving full structure
d session
  │   ├─ calc.ts           # from Chunk 1 (unchanged except integrated into outp
ut object)
  │   └─ storage.ts        # read/write/list structured sessions
  ├─ test/
  │   ├─ calc.test.ts      # unchanged from Chunk 1
  │   └─ storage.test.ts   # NEW tests for full structured session saving + listin
g
  ├─ data/
  │   └─ sessions.json     # auto-created
  ├─ package.json
  ├─ tsconfig.json
  ├─ .gitignore
  └─ README.md
```

Implementation Details

1. Update `storage.ts`

Export the updated interface:

```
export interface Session {
  date: string;
```

```
  exerciseName: string;
  exerciseType: string;
  sets: number;
  weight: number;
  reps: number;
  estimated1RM: number;
  method: string; // "epley"
}
```

Implement/adjust functions:

- `loadSessions()` → creates file if missing, resets corrupt data to `[]`.
- `saveSession(session: Session)` → appends a new entry.
- `listSessions(limit?: number)` → returns most recent first.

2. Update `index.ts` CLI

Parse flags:

- `-sets <n>`
- `-exercise <string>`
- `-equipment <string>`
- `-date <ISO or YYYY-MM-DD>`
- `-save`, `-list [n]`, `-json` (as defined above).

Behavior:

- After computing `estimated1RM`, build a full `Session` object with all 6 raw inputs + calculated fields.
- On `-save`, append this object to `sessions.json` using `saveSession`.
- On `-list [n]`, print recent entries in table form by default, or JSON when `-json` is present.

Example commands:

```
node dist/index.js 225 5 --sets 3 --exercise bench_press --equipment barbell
--save
node dist/index.js --list 10
node dist/index.js --list 5 --json
```

Tests (Vitest)

Create or update `test/storage.test.ts` with tests that:

1. **Creates structured file & loads empty array** when `sessions.json` is missing.
2. **Saves a fully structured session** and verifies that all 6 inputs plus 1RM/method are stored.
3. **Lists sessions correctly (most recent first)** and respects `limit`.
4. **Validates required fields** (`sets`, `exerciseName`, `exerciseType`, date) and throws or handles errors appropriately.
5. **Handles corrupt JSON resets** by rewriting `sessions.json` to `[]` and continuing without crashing.
6. **CLI listing tests** for both table and JSON formats using `-list` and `-list --json`.
7. **CLI saving tests** that confirm full structured objects land in `sessions.json` after `--save`.

All tests must pass via `npm test`.

README.md

Update with:

- Explanation of the new raw data fields.
- Usage examples like:

  ```
  node dist/index.js 185 3 --sets 4 --exercise incline_smith --equipment ma
  chine --save
  ```

```
node dist/index.js --list
node dist/index.js --list 3 --json
```

- JSON schema example of a full `Session` object.
- Notes about date handling, default behavior, and validation rules.

Git Protocol

1. Create a new branch:

   ```
   git checkout -b feat/chunk-2-raw-data-collection
   ```

2. Use Conventional Commits, for example:
   - `feat(cli): add sets, exerciseName, exerciseType, date flags`
   - `feat(storage): store structured session objects`
   - `test(storage): add structured session tests`
   - `docs(readme): document raw data inputs`
3. Merge into `main` after tests pass and tag the release:

   ```
   git tag -a v0.2.0 -m "Chunk 2: raw data collection"
   git push --tags
   ```

Commands to Run

```
npm run build
npm test

# Save a structured session
node dist/index.js 225 5 --sets 3 --exercise bench_press --equipment barbell
--save

# List past sessions (table)
```

```
node dist/index.js --list

# List past sessions (JSON)
node dist/index.js --list 2 --json
```

Acceptance Criteria for Chunk 2

- All 6 input fields (reps, weight, sets, exerciseName, exerciseType, date) are captured and stored correctly.
- `sessions.json` holds fully structured `Session` objects that match the defined schema.
- CLI accepts and correctly parses the new flags for sets, exerciseName, exerciseType, and date.
- Listing works in both table and JSON formats and respects the `limit` parameter.
- Error handling works for missing/invalid fields and corrupt JSON files.
- All Vitest tests for both `calc` and `storage` pass via `npm test`.
- Git history is clean, committed on a feature branch, merged, and tagged `v0.2.0`.

Stop after this. Do **not** implement bodyweight, fatigue/injury factor, trends, or predictions yet. Proceed to Chunk 3 only when data collection is fully working and tested.

Cursor Prompt — Chunk 3: Weekly Max Set Detection (with tests & git)

You are Cursor. Extend the existing **raw data collection system (Chunk 2)** by adding a new feature: **automatically determining the weekly top set and weekly estimated 1RM** based on all logged training sessions. This is **Chunk 3** — do not implement later chunks yet.

This chunk uses the dataset collected in Chunk 2 to produce **weekly summaries** like this:

Week	Date Range	Top Weight	Reps	est1RM
43	Oct 20–26	225	5	262
42	Oct 13–19	215	6	258
41	Oct 06–12	205	8	257

Your job is to implement CLI commands and logic that generate exactly this type of weekly summary table.

Requirements

- **Language/Stack:** Node.js + TypeScript (continue from previous chunks).
- **Goal:** Analyze saved training sessions and compute weekly top sets + weekly estimated 1RM.
- **Target Exercise:** *Only include sessions where* `exerciseType === "barbell"` *and* `exerciseName` *contains "bench".*
- **Weekly Grouping:**
  - Use **ISO Week Number** (weeks start Monday).
  - Compute week start/end dates automatically.
- **Definition of Weekly Top Set:**
  1. Select all bench press / barbell sessions within the same ISO week.
  2. Choose the set with the **heaviest weight**.
  3. If multiple sets share the same weight → pick the one with **fewer reps**.
- **Epley Formula** (reuse Chunk 1 logic):

```
est1RM = weight * (1 + reps/30)
```

- **Output fields:**
  - Week number (e.g., `43`)
  - Date range (e.g., `Oct 20–26`)
  - Top weight

- Reps
- Estimated 1RM

New CLI Features

Add the command:

```
node dist/index.js --weekly
```

Optional flags:

- `-json` → output weekly summary as JSON
- `-limit <n>` → show last `n` weeks only

Examples:

```
node dist/index.js --weekly
node dist/index.js --weekly --limit 5
node dist/index.js --weekly --json
```

Project Scaffolding

Add a new module for analysis:

```
1rm-calculator/
  ├─ src/
  │   ├─ weekly.ts         # NEW: logic for weekly grouping + top set detection
  │   ├─ storage.ts        # existing
  │   ├─ calc.ts           # existing
  │   └─ index.ts          # extend CLI with --weekly
  ├─ test/
  │   ├─ weekly.test.ts    # NEW: tests for weekly summary logic
  │   ├─ storage.test.ts   # existing
  │   └─ calc.test.ts      # existing
```

```
└── data/
    └── sessions.json
```

Implementation Details

1. `src/weekly.ts`

Implement and export:

```
export interface WeeklySummary {
  week: number;
  startDate: string;
  endDate: string;
  topWeight: number;
  reps: number;
  estimated1RM: number;
}

export function getWeeklySummaries(sessions: Session[]): WeeklySummary
[];
```

Logic:

1. Filter sessions:
   - `exerciseType === "barbell"`
   - `exerciseName` contains any of: `"bench"`, `"bench_press"`, `"barbell_bench"`.
2. Group by ISO week.
3. Inside each week:
   - pick the heaviest `weight`
   - if tie: pick the **lowest** `reps`
4. Compute weekly 1RM using Epley.
5. Format week start/end as `"Oct 20–26"`.

6. Sort results by **week number descending**.

2. `src/index.ts`

Add support for:

```
--weekly
--limit <n>
--json
```

- If `-weekly` is used → do not compute a single-set 1RM.
- Instead load `sessions.json`, call `getWeeklySummaries()`, and print table/JSON.

Table Output Example:

```
Week | Date Range     | Top Weight | Reps | est1RM
-----------------------------------------------
43   | Oct 20–26      | 225        | 5    | 262
42   | Oct 13–19      | 215        | 6    | 258
41   | Oct 06–12      | 205        | 8    | 257
```

Tests (Vitest)

Create `test/weekly.test.ts` to test:

1. **Filtering** → only barbell bench press sessions are included.
2. **ISO week grouping** → sessions fall into correct week numbers.
3. **Top set selection** → heaviest weight wins; ties broken by lowest reps.
4. **Date range calculation** → correct Monday–Sunday range.
5. **Epley formula** → weekly 1RM matches Chunk 1 logic.
6. **Sorting** → weeks in descending order.
7. **CLI tests (optional)** → ensure `-weekly` prints correct structure.

All tests must pass.

README.md

README Rules for All Future Chunks (Including Chunk 3):

- **On the feature branch for this chunk (`feat/chunk-3-weekly-max`)** → The README must contain **only the section for Chunk 3**. Do NOT include Chunk 1 or Chunk 2 here.

  Example structure on the feature branch:

  ```
  ## Chunk 3 — Weekly Max Set Detection
  (description, usage, tests, examples)
  ```

- **When merging into `main`** → Append the Chunk 3 section **below the existing Chunk 1 and Chunk 2 sections**. Do NOT overwrite or delete previous sections.

  Example structure on `main` after merge:

  ```
  ## Chunk 1 — Minimal 1RM Calculator
  ...

  ## Chunk 2 — Raw Data Collection
  ...

  ## Chunk 3 — Weekly Max Set Detection
  ...
  ```

These rules ensure:

- Feature branches stay focused and clean.
- The `main` branch becomes a complete chronological project history.

Proceed with adding the Chunk 3 section following this structure.

Do NOT replace the existing README.

Append a new section instead:

Add a section titled:

```
## Chunk 3 — Weekly Max Set Detection
```

This section must be added *after* the Chunk 1 and Chunk 2 sections already in the README. Do not modify or delete earlier chunks.

Include:

Add a new section:

> Weekly Max Set Analysis (Chunk 3)

Include:

- Explanation of how ISO weeks work.
- Description of heaviest-set logic.
- Example usage:

```
node dist/index.js --weekly
node dist/index.js --weekly --limit 4
node dist/index.js --weekly --json
```

- Example output table.

Git Protocol

1. Create a new branch:

   ```
   git checkout -b feat/chunk-3-weekly-max
   ```

2. Conventional commits:

   - `feat(weekly): add ISO week grouping and top-set detection`
   - `feat(cli): add --weekly summary output`
   - `test(weekly): add weekly summary tests`
   - `docs(readme): document weekly analysis feature`

3. Merge and tag:

```
git tag -a v0.3.0 -m "Chunk 3: Weekly max set analysis"
git push --tags
```

Commands to Run

```
npm run build
npm test

# Generate weekly max sets     node dist/index.js --weekly

# Limit to last 4 weeks
node dist/index.js --weekly --limit 4

# JSON output
node dist/index.js --weekly --json
```

Acceptance Criteria for Chunk 3

- Weekly top sets are computed correctly using your raw logged data.
- Only barbell bench press sessions are included.
- Weekly summaries include: week number, date range, top weight, reps, est1RM.
- Sorting, filtering, and tie-breaking rules follow the Notion/PDF spec.
- Tests verify grouping, selection, and output correctness.
- Git branch/commit/tag structure is clean.

Stop after this. Do **not** implement trends, injury factors, prediction, or modeling yet. Chunk 4 begins once weekly analysis is fully working.

Cursor Prompt — Chunk 4: Bodyweight & Relative Strength (with tests, git safety, and README rules)

You are Cursor. Implement **Chunk 4** of the 1RM prediction project. Extend the system built in **Chunks 1–3** by adding **bodyweight tracking** and **relative strength analysis**. Follow all safety rules, branching rules, and README rules described below.

Do **not** implement later chunks yet.

Goal

Chunk 4 introduces two major enhancements:

1. **Bodyweight logging** for each saved session.
2. **Relative strength calculation:**

   ```
   relativeStrength = estimated1RM / bodyweight
   ```

3. Optional classification of relative strength (e.g., novice/intermediate/advanced/elite).
4. New CLI command `-rel` to inspect and analyze bodyweight + relative strength data.

This continues to build the dataset needed for later model-based predictions.

Requirements

New Input Field

- `-bw <number>` → user's bodyweight at time of session

Bodyweight must be:

- A positive number
- Optionally omitted (in which case relative strength will be `null`)

New Derived Fields

Extend each stored session to include:

```
bodyweight?: number | null;
relativeStrength?: number | null;
strengthCategory?: string | undefined;  // e.g., novice/intermediate/advanced/
elite
```

Relative Strength Rules

- If bodyweight is provided and > 0:
  - Compute `relativeStrength = estimated1RM / bodyweight`
- If absent or invalid → leave both fields `null`

Strength Categories (simple heuristic)

```
< 1.0 → novice
1.0–1.49 → intermediate
1.5–1.99 → advanced
≥ 2.0 → elite
```

CLI Features

1. Saving bodyweight data

```
node dist/index.js 225 5 --sets 3 --exercise bench_press --equipment barbell
--bw 180 --save
```

This should save:

- weight, reps, sets, exerciseName, exerciseType
- estimated1RM (Chunk 1)
- bodyweight (Chunk 4)

- relativeStrength + strengthCategory (Chunk 4)

2. Relative strength analysis mode

```
node dist/index.js --rel
```

This prints a table sorted by most recent sessions:

```
date       exercise      weight reps est1RM bw   rel  category
2025-10-20  bench_press   225    5    263   180  1.46 intermediate
```

3. JSON mode

```
node dist/index.js --rel --json
```

Outputs objects including:

```
{
  "date": "2025-10-20T00:00:00Z",
  "exerciseName": "bench_press",
  "exerciseType": "barbell",
  "weight": 225,
  "reps": 5,
  "estimated1RM": 263,
  "bodyweight": 180,
  "relativeStrength": 1.46,
  "strengthCategory": "intermediate"
}
```

File Changes

Modify `storage.ts`

- Update the `Session` interface

- Ensure `loadSessions()` supports older sessions gracefully

Update `index.ts`

- Add parsing of `-bw`
- Add `-rel` mode
- Integrate relative strength logic into existing save path

New helper file (recommended): `src/rel.ts`

```
export function computeRelativeStrength(estimated1RM: number, bw?: numbe
r | null): number | null
export function classifyStrengthRatio(ratio: number | null): string | undefined
```

Tests (Vitest)

Create a new file: `test/rel.test.ts`:

1. `computeRelativeStrength` returns accurate division
2. Returns `null` for missing or invalid bodyweight
3. Category classification matches rules

Update `test/storage.test.ts`:

- Saving a session writes bodyweight & relative strength
- Old sessions still load without crashing

Optional CLI test:

- Spawn CLI with `-rel --json` and assert correct fields

All tests must pass.

README.md Rules (IMPORTANT)

On the feature branch (`feat/chunk-4-bodyweight-rel`)

`README.md` must contain **only:**

```
## Chunk 4 — Bodyweight & Relative Strength
(description, usage examples, explanation of fields)
```

Do **not** include Chunk 1–3 here.

On `main` branch (AFTER merging)

Append the Chunk 4 section below the existing ones:

```
## Chunk 1 — ...
## Chunk 2 — ...
## Chunk 3 — ...
## Chunk 4 — Bodyweight & Relative Strength
```

Never overwrite previous chunks.

Git Protocol (Safe, Required)

1. Create feature branch

```
git checkout -b feat/chunk-4-bodyweight-rel
```

2. After implementing & testing:

Commit changes atomically:

```
git add .
git commit -m "feat(rel): add bodyweight and relative strength fields"
git commit -m "feat(cli): add --bw and --rel commands"
git commit -m "test(rel): add tests for relative strength and categories"
git commit -m "docs(readme): add chunk 4 section for bodyweight & relative s
trength"
```

3. Push the branch (safety)

```
git push -u origin feat/chunk-4-bodyweight-rel
```

4. Manual merge to main (do NOT automate)

```
git checkout main
git merge feat/chunk-4-bodyweight-rel
git push origin main
```

Commands to Run

```
npm run build
npm test

node dist/index.js 225 5 --sets 3 --exercise bench_press --equipment barbell
--bw 180 --save
node dist/index.js --rel
node dist/index.js --rel --json
```

Acceptance Criteria

- Bodyweight saved correctly
- Relative strength and category computed correctly
- `-rel` outputs table + JSON
- Old sessions still load
- All tests pass
- README updated per branch rules
- Branch pushed safely

Stop here. Do not implement future chunks yet.

Cursor Prompt — Chunk 5: Fatigue & Recovery Index (with tests, git safety, README rules)

You are Cursor. Implement **Chunk 5** of the project by adding a **fatigue/recovery factor** to each session.

Do NOT implement future chunks yet.

Goal

Chunk 5 introduces session-level training modifiers:

- **Fatigue score** (`-fatigue <0–10>`)
- **Recovery score** (`-recovery <0–10>`)
- Derived metric: `adjusted1RM = estimated1RM * recoveryModifier - fatiguePenalty`

These modifiers improve the model's understanding of performance fluctuations.

Requirements

New Inputs

- `-fatigue <0–10>`
- `-recovery <0–10>`

Derived Fields

Store these inside each session:

```
fatigue?: number | null;
recovery?: number | null;
adjusted1RM?: number | null;
```

Rules

- If fatigue or recovery missing → leave fields as `null`.

- Recovery modifier (example): `1 + (recovery - 5) * 0.02`
- Fatigue penalty (example): `fatigue * 0.5`
- Adjusted 1RM:

```
adjusted1RM = estimated1RM * recoveryModifier - fatiguePenalty
```

All formulas are placeholders and can be tuned later.

CLI Features

```
node dist/index.js 225 5 --sets 3 --exercise bench_press --equipment barbell --bw 180 --fatigue 6 --recovery 4 --save
```

Adds fatigue/recovery to stored session.

New display command:

```
node dist/index.js --fatigue-trend
```

Outputs a table of sessions sorted by date with fatigue, recovery, adjusted1RM.

- `-json` mode supported.

Tests

Add `test/fatigue.test.ts`:

- Validate fatigue/recovery ranges (0–10)
- adjusted1RM formula correctness
- Missing fields → null

Update `storage.test.ts`:

- Sessions saved/loaded correctly with new fields
- Old sessions load without failure

README Rules

Feature Branch README

Contains ONLY:

```
## Chunk 5 — Fatigue & Recovery Index
```

Main Branch README

Append Chunk 5 section after Chunk 4.

Git Protocol

```
git checkout -b feat/chunk-5-fatigue-recovery
# implement
# commit with Conventional Commits
git push -u origin feat/chunk-5-fatigue-recovery
```

Manual merge into main after testing.

Cursor Prompt — Chunk 6: Long-Term Trend Analysis (with tests, git safety, README rules)

Implement **Chunk 6**: detect long-term strength trends.

Goal

Build a module that analyzes changes in weekly estimated 1RM over time.

This will support modeling in Chunks 7–8.

Requirements

New CLI Command

```
node dist/index.js --trend
```

Outputs:

- Week number
- est1RM
- 3-week moving average
- 5-week moving average
- Rate of change (delta)

JSON mode supported.

Data Source

Uses weekly summaries from Chunk 3.

File Changes

Create `src/trend.ts`:

```
export function computeMovingAverage(values: number[], window: number):
number[];
export function computeTrend(weeklySummaries: WeeklySummary[]): TrendP
oint[];
```

TrendPoint:

```
interface TrendPoint {
  week: number;
  est1RM: number;
  ma3: number | null;
  ma5: number | null;
```

```
  delta: number | null;
}
```

Tests

`test/trend.test.ts`:

- MA calculations correct
- Delta correct
- Weekly ordering correct

README Rules

Feature branch README → only Chunk 6.

Main README → append Chunk 6.

Git Protocol

```
git checkout -b feat/chunk-6-long-term-trends
git push -u origin feat/chunk-6-long-term-trends
```

Manual merge later.

Cursor Prompt — Chunk 7: Model Training & Prediction (with tests, git safety, README rules)

Implement **Chunk 7**: build a lightweight prediction model using collected data.

Goal

Use all stored features:

- weight, reps, sets
- bodyweight
- relativeStrength
- fatigue/recovery
- weekly trends

  To train a **simple regression model** to predict true 1RM.

No neural networks yet — keep it simple.

Requirements

Model Type

Use a simple linear regression:

```
pred1RM = b0 + b1*weight + b2*reps + b3*bodyweight + b4*relativeStrength
+ b5*fatigue + ...
```

CLI

```
node dist/index.js --train
node dist/index.js --predict --weight 225 --reps 4 --bw 180
```

Model saved to:

```
data/model.json
```

File Changes

Create `src/model.ts`:

- `trainModel(sessions)` → returns coefficients
- `predict(model, inputs)` → returns predicted 1RM

Tests

`test/model.test.ts`:

- Training with dummy data yields correct coefficients
- Prediction works for known inputs

README Rules

Feature branch → only Chunk 7 README.

Main branch → append Chunk 7.

Git Protocol

```
git checkout -b feat/chunk-7-model-training
git push -u origin feat/chunk-7-model-training
```

Cursor Prompt — Chunk 8: Final Integration, UI Polish & Release (with tests, git safety, README rules)

Implement **Chunk 8**, the final chunk.

Goal

- Validate all features end-to-end
- Improve CLI messages
- Add help command
- Add version command
- Prepare final release

Requirements

New CLI Features

```
node dist/index.js --help
node dist/index.js --version
```

Help prints all commands from Chunks 1–7.

Version reads from `package.json`.

Code cleanup

- Remove unused functions
- Consolidate utilities
- Ensure TypeScript strict mode passes

Release

- Increment version in `package.json`
- Build final binary/scripts
- Tag release

README Rules

Feature branch → only Chunk 8 README.

Main branch → append as final section.

Git Protocol

```
git checkout -b feat/chunk-8-final-release
git push -u origin feat/chunk-8-final-release
```

After testing: merge into main, tag `v1.0.0`.You are Cursor. Implement **Chunk 4** of the 1RM prediction project. Extend the system built in **Chunks 1–3** by adding

bodyweight tracking and **relative strength analysis**. Follow all safety rules, branching rules, and README rules described below.

Do **not** implement later chunks yet.

Goal

Chunk 4 introduces two major enhancements:

1. **Bodyweight logging** for each saved session.
2. **Relative strength calculation:**

   ```
   relativeStrength = estimated1RM / bodyweight
   ```

3. Optional classification of relative strength (e.g., novice/intermediate/advanced/elite).
4. New CLI command `-rel` to inspect and analyze bodyweight + relative strength data.

This continues to build the dataset needed for later model-based predictions.

Requirements

New Input Field

- `-bw <number>` → user's bodyweight at time of session

Bodyweight must be:

- A positive number
- Optionally omitted (in which case relative strength will be `null`)

New Derived Fields

Extend each stored session to include:

```
bodyweight?: number | null;
relativeStrength?: number | null;
```

```
strengthCategory?: string | undefined;  // e.g., novice/intermediate/advanced/elite
```

Relative Strength Rules

- If bodyweight is provided and > 0:
  - Compute `relativeStrength = estimated1RM / bodyweight`
- If absent or invalid → leave both fields `null`

Strength Categories (simple heuristic)

```
< 1.0 → novice
1.0–1.49 → intermediate
1.5–1.99 → advanced
≥ 2.0 → elite
```

CLI Features

1. Saving bodyweight data

```
node dist/index.js 225 5 --sets 3 --exercise bench_press --equipment barbell --bw 180 --save
```

This should save:

- weight, reps, sets, exerciseName, exerciseType
- estimated1RM (Chunk 1)
- bodyweight (Chunk 4)
- relativeStrength + strengthCategory (Chunk 4)

2. Relative strength analysis mode

```
node dist/index.js --rel
```

This prints a table sorted by most recent sessions:

```
date       exercise      weight reps est1RM bw   rel  category
2025-10-20  bench_press   225    5    263   180  1.46 intermediate
```

3. JSON mode

```
node dist/index.js --rel --json
```

Outputs objects including:

```
{
  "date": "2025-10-20T00:00:00Z",
  "exerciseName": "bench_press",
  "exerciseType": "barbell",
  "weight": 225,
  "reps": 5,
  "estimated1RM": 263,
  "bodyweight": 180,
  "relativeStrength": 1.46,
  "strengthCategory": "intermediate"
}
```

File Changes

Modify `storage.ts`

- Update the `Session` interface
- Ensure `loadSessions()` supports older sessions gracefully

Update `index.ts`

- Add parsing of `-bw`
- Add `-rel` mode

- Integrate relative strength logic into existing save path

New helper file (recommended): `src/rel.ts`

```
export function computeRelativeStrength(estimated1RM: number, bw?: numbe
r | null): number | null
export function classifyStrengthRatio(ratio: number | null): string | undefined
```

Tests (Vitest)

Create a new file: `test/rel.test.ts`:

1. `computeRelativeStrength` returns accurate division
2. Returns `null` for missing or invalid bodyweight
3. Category classification matches rules

Update `test/storage.test.ts`:

- Saving a session writes bodyweight & relative strength
- Old sessions still load without crashing

Optional CLI test:

- Spawn CLI with `-rel --json` and assert correct fields

All tests must pass.

README.md Rules (IMPORTANT)

On the feature branch (`feat/chunk-4-bodyweight-rel`)

`README.md` must contain **only:**

```
## Chunk 4 — Bodyweight & Relative Strength
(description, usage examples, explanation of fields)
```

Do **not** include Chunk 1–3 here.

On `main` branch (AFTER merging)

Append the Chunk 4 section below the existing ones:

```
## Chunk 1 — ...
## Chunk 2 — ...
## Chunk 3 — ...
## Chunk 4 — Bodyweight & Relative Strength
```

Never overwrite previous chunks.

Git Protocol (Safe, Required)

1. Create feature branch

```
git checkout -b feat/chunk-4-bodyweight-rel
```

2. After implementing & testing:

Commit changes atomically:

```
git add .
git commit -m "feat(rel): add bodyweight and relative strength fields"
git commit -m "feat(cli): add --bw and --rel commands"
git commit -m "test(rel): add tests for relative strength and categories"
git commit -m "docs(readme): add chunk 4 section for bodyweight & relative s
trength"
```

3. Push the branch (safety)

```
git push -u origin feat/chunk-4-bodyweight-rel
```

4. Manual merge to main (do NOT automate)

```
git checkout main
git merge feat/chunk-4-bodyweight-rel
git push origin main
```

Commands to Run

```
npm run build
npm test

node dist/index.js 225 5 --sets 3 --exercise bench_press --equipment barbell
--bw 180 --save
node dist/index.js --rel
node dist/index.js --rel --json
```

Acceptance Criteria

- Bodyweight saved correctly
- Relative strength and category computed correctly
- `-rel` outputs table + JSON
- Old sessions still load
- All tests pass
- README updated per branch rules
- Branch pushed safely

Stop here. Do not implement future chunks yet.