**M7 – Megadetaljert gjennomføringsplan**

**Blokk 1 – Forberedelser og oppsett**

1. **Oppdater lokal repo**

bash

KopierRediger

git pull origin main

Sjekk at du har siste masterplan og kildekode.  
Hvis du får konflikt, løs den før du starter.

1. **Lag egen branch for M7**

bash

KopierRediger

git checkout -b feature/m7-analysis

Dette sikrer at du kan jobbe uten å påvirke main før alt er testet.

1. **Sjekk at Python og Rust-miljøet fungerer**
   * Test Maturin-build:

bash

KopierRediger

maturin develop

* + Importer i Python REPL:

python

KopierRediger

import cyclegraph\_core

Ingen feil → alt OK.

**Blokk 2 – Implementere kjerneberegninger i Rust (NP, IF, VI, Pa:Hr, W/beat)**

1. Åpne src/lib.rs (eller tilsvarende kjernefil for Rust-logikken).
2. Lag funksjon **analyze\_session** med signatur fra DOD:

rust

KopierRediger

pub fn analyze\_session(samples: Samples, meta: Meta, cfg: Config) -> SessionReport

1. Implementer:
   * **NP**: 30s rullende snitt, ^4-middel, deretter ^(1/4).
   * **IF**: np / ftp. FTP fallback: beste 20 min × 0,95 siste 30–60 dager.
   * **VI**: np / avg\_power.
   * **Pa:Hr**: Del økten i to halvdeler, beregn P/HR-ratio, finn relativ endring.
   * **W/beat**: avg\_power / avg\_hr, normalisert mot baseline (median siste 28 dager ±25 % varighet).
2. **Feilsøking underveis:**
   * Kjør cargo test ofte for å fange tidlige feil.
   * Bruk syntetiske testserier for å sjekke at NP/IF/VI-formlene gir riktige tall.

**Blokk 3 – CGS v1 logikk**

1. Implementer CGS-score:
   * Intensitet 40 %: formelen fra DOD (100 \* exp(...)).
   * Varighet 30 %: log-skalert poeng basert på minutter.
   * Kvalitet 30 %: kombinasjon av DriftScore, VIScore, WpBScore.
2. Legg til fallback-logikk:
   * Hvis en komponent mangler → revekt de andre.
   * Hvis alt mangler → default 50.
3. Legg inn stykkevise skalaer for Pa:Hr, VI, W/beat (fra tidligere spesifikasjon).
4. Lag trend-felt for siste 3 (og evt. 5) økter.

**Blokk 4 – Badges og regelmotor**

1. Implementer første versjon av badges:
   * Iron Lungs → Pa:Hr ≤ 2 % på ≥ 90 min.
   * Metronome → VI ≤ 1.03 på ≥ 60 min.
   * Big Engine → W/beat ≥ +10 % baseline.
2. Sørg for at badges legges til i SessionReport som en strengliste.

**Blokk 5 – API og CLI**

1. I Rust (lib.rs):
   * analyze\_batch for trendberegning.
   * estimate\_ftp for FTP-estimat.
2. I Python-CLI (analyze.py):
   * Legg til flagg:

diff

KopierRediger

--input data/\*.csv

--out output/

--cfg config.json

--batch

--with-trend

--set-ftp

--auto-ftp

--format json|csv|both

--publish-to-strava

* + Implementer --publish-to-strava (les siste importerte aktivitet, oppdater kommentar/beskrivelse).

1. **Tørrkjøring:**  
   --dry-run skal kun skrive til stdout uten å publisere.

**Blokk 6 – Robusthet og tester**

1. **Unit-tester** i Rust:
   * NP/IF/VI/Pa:Hr/WpB (syntetiske data).
2. **Golden-tester**:
   * 3 ekte økter (rolig langtur, terskel, intervaller).
3. **Edge-tester**:
   * Uten watt
   * Uten HR
   * Korte økter (<10/20/60 min)
   * Outliers
4. Legg inn meningsfulle feilmeldinger i CLI med tydelige exitkoder.

**Blokk 7 – Dokumentasjon og commit**

1. Skriv:
   * docs/CGS\_v1.md (formler, skalaer, fallbacks)
   * docs/cli\_usage.md (eksempler)
   * docs/strava\_publish.md (publisering + av/på)
2. Commit i riktig rekkefølge (fra DOD):

bash

KopierRediger

git add .

git commit -m "feat(m7): core CGS v1 + metrics (NP/IF/VI/Pa:Hr/WpB) with pyo3"

git commit -m "feat(m7): batch trend + badges + JSON schema"

git commit -m "feat(m7): CLI flags + Strava auto-publish (+dry-run)"

git commit -m "test(m7): unit/golden/edge + perf guard"

1. Push branch og opprett PR:

bash

KopierRediger

git push origin feature/m7-analysis

✅ **Ferdigkriterier for M7:**

* Alle beregninger (NP, IF, VI, Pa:Hr, W/beat) og CGS v1 virker.
* Trend og badges fungerer.
* CLI har alle flagg, inkl. Strava-publisering.
* Alle tester (unit, golden, edge) er grønne.
* Ytelse: 2 t @ 1 Hz ≤ 200 ms på laptop.
* Dokumentasjon ferdig og i repo.
* Masterplan-status for M7 satt til *Ferdig*.

DET som står igjen 20%

**M7 “to‑do” (kort, konkret – siste 20 %)**

**1) WpB‑baseline (28 dager) + badge**

**Mål:** beregn baseline (median) av w\_per\_beat for forrige 28 dager (match ~±25 % varighet mot økta), sett w\_per\_beat\_baseline i rapporten og aktiver “Big Engine”.

**Minste løsning nå (CLI‑nivå):**

* Lag en enkel **historikk‑mappe** (JSON‑rapporter du allerede genererer).
* Når du kjører session --batch, last inn siste 28 dager for samme bruker (filnavnkonvensjon), filtrer økter med varighet innen ±25 % av aktuelle økt, beregn median av w\_per\_beat → sett som baseline for hver ny rapport i batchen.
* Aktiver badge: if (w - b)/b >= 0.10: "Big Engine".

**Akseptanse:**

* w\_per\_beat\_baseline er ikke null når historikk finnes.
* “Big Engine” dukker korrekt opp/ikke opp avhengig av baseline.

**2) Trend i Rust (valgfritt nå) eller behold i CLI**

**Mål:** trendfelt for siste 3 økter (cgs\_last3\_avg, cgs\_delta\_vs\_last3).

**Nåværende:** gjort i Python (--with-trend).  
**Ok for M7:** behold slik, men dokumenter det i docs/cli\_usage.md.

**3) Tester (unit/golden/edge)**

**Mål:** sikre beregninger + robusthet.

* **Rust unit** (i core/tests/):
  + Konstant serie (NP≈avg, VI≈1.0, Pa:Hr≈0).
  + Intervallserie med høy VI.
  + Kort økt (<30s) → np=None.
* **Golden** (CLI): 3 CSV (rolig, terskel, intervall) med forventningssjekk (np/vi/pa\_hr intervall).
* **Edge**:
  + Uten watt → np=None, vi=None, men HR‑felt finnes.
  + Uten HR → pa\_hr\_pct=None, w\_per\_beat=None.
  + Outliers: verdier > 1500 W eller HR > 220 → ignorer/klipp i parser (CLI).

**Akseptanse:** cargo test grønn; egen tests\_golden.ps1 som returnerer exit‑kode 0.

**4) Dokumentasjon**

Lag tre korte dokumenter i docs/:

* CGS\_v1.md: formler (NP/IF/VI/Pa:Hr/WpB), skalaer, badges, baseline‑logikk.
* cli\_usage.md: hvordan kjøre efficiency og session med eksempler (inkl. batch, trend, auto‑FTP).
* strava\_publish.md: plan/krav for M8 (API‑scope, hva som publiseres, “dry‑run”).

**Akseptanse:** alle tre filer finnes, lettleste, og matcher faktisk oppførsel.

**5) Commits & PR**

Kjør fra repo‑roten:

powershell

KopierRediger

git add .

git commit -m "feat(m7): core metrics (NP/IF/VI/Pa:Hr/WpB) + CGS v1 + Api.analyze\_session\_json"

git commit -m "feat(cli): session subcommand with batch/trend/auto-ftp/format/dry-run + efficiency retained"

git commit -m "test(m7): unit/golden/edge coverage + basic perf guard"

git commit -m "docs(m7): CGS\_v1, cli\_usage, strava\_publish"

git push origin feature/m7-analysis

# Opprett PR mot main

**Mini‑referat (kopiér til ny chat)**

* Rust‑kjerne bygget (pyo3 0.22.x) og installert med maturin develop. API tilgjengelig via cyclegraph\_core.Api.
* Implementert: NP, IF, VI, Pa:Hr, W/beat; CGS v1 (intensitet/varighet/kvalitet); badges: Iron Lungs, Metronome.
* CLI (cli/analyze.py) har to subcommands:
  + efficiency (watt/puls + SHACL) – **fixet CSV‑parser**.
  + session (M7): batch, minitrend (i CLI), auto‑FTP (20min\*0.95), JSON/CSV, dry‑run; Strava = stub.
* Gjenstår for M7: WpB‑baseline 28d (+ Big Engine), enkle tester (unit/golden/edge), docs, commits/PR.

Hvis du vil, kan jeg i neste chat levere:

* ferdig baseline\_history.py hjelpe­modul + endring i session som finner 28d baseline og aktiverer “Big Engine”.
* en liten tests\_golden.ps1 som kjører 3 CSV’er og sjekker forventninger.

Spør ChatGPT