**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*.

**Neste steg**

**4.3 – Strava klient**

Fil: cli/strava\_client.py  
Mål: Publisere COMMENT og DESC fra formatteren direkte til siste Strava-aktivitet.

**Oppgaver:**

1. Gjenbruk M6 token/refresh-logikk.
2. Implementere:
   * get\_latest\_activity\_id() -> int  
     (Bruk API: /athlete/activities?per\_page=1 for siste aktivitet)
   * create\_comment(activity\_id: int, text: str) -> None  
     (API: POST /activities/{id}/comments)
   * update\_description(activity\_id: int, header: str) -> None  
     (API: PUT /activities/{id})
3. Fallback:
   * Bruk state/last\_import.json hvis tilgjengelig.
   * Ellers hent siste aktivitet via get\_latest\_activity\_id().
4. Robusthet:
   * Håndter 401/403 (token refresh) og 5xx (retry eller feilmelding).
   * Logg tydelig ved rate-limit.
5. Integrasjon:
   * Oppdatere cli/analyze.py til å kalle Strava-klienten når ikke --dry-run.

**Definition of Done (DoD):**

* Kommentar og beskrivelse oppdateres på en testaktivitet på Strava.
* Språkvalg (--lang) respekteres.
* Ingen hengende separatorer eller dobbelt-mellomrom.
* Ingen API-feil ved gyldig token.

**5 – Robusthet & tester**

* **5.1** Python-tester:
  + Formatter-tester er delvis ferdig.
  + Legg til CLI-tester for Strava-publish (mock API-kall).
* **5.2** Rust-tester:
  + Unit-tester for NP, IF, VI, Pa:Hr, W/slag.
  + Edge-tester (ingen watt/HR, korte økter, pauser, outliers).
  + Golden-tester (3 ekte økter, toleranser).
  + Perf-guard: ≤ 200 ms for 2 t @ 1 Hz.

**6 – Dokumentasjon**

* docs/CGS\_v1.md → Formler, skalaer, fallbacks, badges, ytelse.
* docs/cli\_usage.md → Eksempler, flagg, formater, Strava-publisering.
* docs/strava\_publish.md → Tekstformat, språkvalg, trimming, krav.

**7 – Comms & commits**

* Ryddige commits pr. funksjon/fil.
* Oppdatere Masterplan når alle oppgaver er “Ferdig”.
* Loggføre sluttstatus i statusoversikt.

**Kommandoer for testing når 4.3 er implementert:**

bash

KopierRediger

# Dry-run (ingen Strava-posting)

python -m cli.analyze session --input "data/streams/\*.csv" --out output --cfg config.json --format json --lang no --dry-run

# Ekte publisering

python -m cli.analyze session --input "data/streams/\*.csv" --out output --cfg config.json --format json --lang n