**Workflow – CycleGraph utviklingsflyt**

**Steg 1 – Produser Manus til Copilot**

**KJØRING  
Implementer Sprint 2 Vær & profiler (🌤️) i CycleGraph.`**

**Du skal nå gå gjennom sprinten steg for steg sammen med meg, og følge arbeidsmetoden under.**

**Arbeidsmetode**

1. **Foreslå 5–7 små trinn.**
2. **For hvert trinn:  
   a) hvilke filer + hvorfor (1–2 setn) · b) forventet output (tester/CLI) · c) vent “OK” · d) vis diff, kjør tester, rapporter.**
3. **Feil → stopp, minimer scope, enkel fix først.**
4. **Ingen store refaktorer, ikke rør “Ikke rør”, én commit per trinn.**
5. **Ferdig når alle DoD er verifisert med tester/CLI.**

**DoD (må alle bestås)**

**1.- Implementere wattberegning v1 med værdata og sykkeltype som input.**

**2.- Utvide Rust-funksjonene, teste med golden tests + live økter.**

**3. Perf-sjekk (idempotens, kaldstart < 3s).**

**4. Oppdater CGS score til v1.1 når implementasjonen fungerer og er testet.**

**NBNB vi skal ikke gjøre noe med Cli eller Frontend i denne sprinten det kommer senere.**

**HUSK: Git hygiene: .gitignore skjuler secrets/state; eksempelfiler (sample.json / \*\_example.py) ligger i repo, uten sensitive data.- “Repo-rydding planlagt: .gitignore for secrets/state + eksempelfiler for tokens/state.”).Sprint-delta (mini-kontekst)**

**Endringsgrenser (HALVFAST DEFAULT)**

* **Tillatte: core/src/lib.rs, core/src/metrics.rs, cli/\*.py, tests/\*\*, .gitignore, state/\*.sample.json, cli/\*\_example.py**
* **Ikke rør: Watt-engine terskler/beregninger, CI-workflow, eksterne API-kontrakter**

**Tester / kommandoer (HALVFAST DEFAULT)**

* **pytest -q · cargo test -q · python cli/analyze.py --dry-run --activity <id>**

**Verifikasjonsplan (DoD → test) (HALVFAST DEFAULT, konkretisert for S2)**

* **(1): pytest repeterte kall på samme (lat,lon,timestamp) ⇒ ≥95 % cache-hit, ingen panic i Rust.**
* **(2): pytest fixture med manglende profilfelt ⇒ defaults settes + estimat=true.**
* **(3): enhetstest/cargo test verifiserer at weather\_cache\_hit\_total / miss\_total teller riktig.**
* **(4): CLI dry-run viser weather-data + profil + estimatfelt i stdout/JSON.**
* **(5): sjekk .gitignore; profile.sample.json og cache-mappe finnes uten sensitive data.**
* **(6)Re-enable Rust-tester – sikrer at kjernen ikke regress’er. Lav risiko, høy verdi.**
* **(7)WD=core/ i CI-steg – enkelt grep som hindrer snublefeil.**

**Forventet output (HALVFAST DEFAULT, konkretisert for S2)**

* **• Filer: core/src/lib.rs, core/src/metrics.rs, cli/analyze.py, tests/\*\* (vær/profil-tester), .gitignore, state/last\_import.sample.json**

**Avslutningskommando (mal):**

LAG EN SLUTTRAPPORT med følgende format:

✅ Sprint: [ID – Navn]

Branch: [navn]

Commits: [hash – message]

Endrede filer: [paths]

Tester: [pytest + cargo test resultater]

Observasjoner: [korte punkter]

Status: [Ferdig / Delvis ferdig / Feil gjenstår]

**🔹 Sluttføring av Workflow**

**1. Sluttrapport**

* Lim inn Copilot sin sluttrapport i Template (/reports/).

**2. Delta Sammendrag**

* Skriv 2–3 linjer pr sprint med komprimeringsmal.
* Lim inn nederst i Temate.md under *Delta Sammendrag av Sluttrapporter*.

📋 **Komprimeringsmal**

1. Hva ble gjort – hovedleveranse/funksjon.
2. Teststatus – oppsummer kort (✅ pytest/cargo grønt, evt. feil).
3. Observasjon – maks én setning (læring, carry-over, viktig oppdagelse).

**3. Evaluering (beslutningsdrevet)**

1. Oppdater DoD og Frontlog:
   * Les siste sluttrapport → identifiser nye funn.
   * Klassifiser funn:
     + **A. Akseptansekriterier/risiko** → må inn i Dynamisk DoD nå.
     + **B. Oppgaver/forbedring** → til Frontlog Kommende Sprinter (ev. åpne SxB-minisprint).
     + **C. Innsikt/observasjon** → kun som kontekst/carry-over i sammendrag.
2. Oppdater Template for neste sprint:
   * Se på forrige Template og vurder endringer i halvfaste felt (kun hvis nødvendig).
   * Sjekk om **Endringsgrenser / Verifikasjonsplan / Forventet output** trenger justeringer.
   * Hvis **JA** → oppdater “HALVFAST DEFAULT”-blokkene (ny default fremover).
   * Hvis **NEI** → behold som er.
3. Fyll inn resten av nytt sprintmanus til Copilot: Sprintnavn, DoD og Sprint-delta (mini-kontekst).
4. Lim inn i Sprintmanus → klart for neste sprint.

**4. Oppdater DoD & Backlog**

* Flytt ferdige sprinter til ✅ Ferdig.
* Marker eventuelle avhengigheter som åpne.

**5. Synk Masterplan**

* Oppdater statuslinje eller legg til ny rad hvis relevant.

**6. Sanity systemtester (før push)**

1. pytest -q og cargo test --tests -q grønne lokalt + CI.
2. **Idempotens:** kjør samme analyse to ganger → diff skal være tom (filtrer tidsstempler).
3. **Golden/determinisme:** toleranse ±1–2 W, ingen flaky tester.
4. **Dokumentasjon:** DoD, Masterplan og Backlog synket. Hvis noe gjenstår → åpne *SxB-minisprint*.
   * Hvis sprint berører Strava-publisering: verifiser tests/test\_strava\_client.py.

**7. Commit & Push**

* Følg navnestandarder:
  + **Branch:** feature/sprint-sx-<kort-navn>  
    (eks: feature/sprint-s1-mode-detection)
  + **Commit-prefix:** Sx:  
    (eks: S1: add --mode roller|outdoor)
  + **Sluttrapport-fil:** YYYY-MM-DD\_sprint-Sx\_<kort-navn>.md

**8. Oppdater Log**

* Kort notat i logs/ med status og ev. carry-over.

**🔹 Eksempel (S1B ferdig → neste S2)**

Neste sprint = **S2 – Vær & profiler**

* Kopiér oppstartsmalen.
* Sett sprint-info til S2.
* Lim inn DoD-utdrag: vær-API, profilsettings, caching, validering.