**Core-Loop Treasury Auto-Sweep — Technical Specification (Buildable by Claude)**

**Scope:** Implement an *offline*, explainable auto-sweep engine using local CSV/JSON inputs that produces a human-readable EOD summary and an optional simulated order.  
**Loop:** **Parse → Predict → Prescribe → Perform → Present**  
**Language:** Python **3.12** (single-process, file-based; no external services required)

**1) Goals & Non-Goals**

**1.1 Goals**

* Deterministic computation of short-horizon cash needs and safe deployable surplus.
* Single-bucket allocation proposal (e.g., “Liquid\_Fund\_Overnight”) with guardrails.
* Clear **reason codes** for every decision (explainability).
* CLI entrypoint that reads CSV/JSON and writes summary.json + EOD\_Summary.md.
* Optional simulated execution writing order\_{uuid}.json.

**1.2 Non-Goals**

* Live connectors (bank/ERP/venues), production infra, portfolio optimization, advanced ML, SOC-grade audit.

**2) Runtime, Dependencies, Commands**

**2.1 Environment**

* Python **3.12**
* OS-agnostic; reads/writes local files.

**2.2 Dependencies**

* **Required:** pandas, pytest
* **Optional:** pydantic (typed models), streamlit (demo UI), rich (pretty logs)

**2.3 Standard commands**

# install

pip install pandas pytest

# optional extras

pip install pydantic streamlit rich

# run CLI end-to-end (default horizon=7)

python -m src.cli --data-dir data --out-dir outputs --horizon 7 --execute

# run tests

pytest -q

**3) Project Layout**

/core-loop

/data/ # INPUTS (CSV/JSON)

bank\_txns.csv

ar\_invoices.csv

ap\_bills.csv

policy.json

cutoff\_calendar.json

/src/

\_\_init\_\_.py

cli.py # orchestration (one-shot run)

/core/

\_\_init\_\_.py

config.py # load\_settings(data\_dir) -> Settings

reason\_codes.py # REASONS = {...}

parse.py # loaders, balance computation

predict.py # AR/AP horizon flows (expected inflows/outflows)

prescribe.py # policy math, allocation, cutoff/maker-checker

perform.py # simulated order lifecycle

present.py # EOD markdown builder

models.py # (optional) pydantic models

/outputs/ # OUTPUTS

summary.json

EOD\_Summary.md

order\_{uuid}.json # if execute & order exists

/tests/

test\_core\_loop.py

README.md

requirements.txt # minimal pin set

**4) Data Contracts (Files → Canonical Shapes)**

**4.1 bank\_txns.csv**

| **column** | **type** | **required** | **notes** |
| --- | --- | --- | --- |
| date | ISO date | yes | posting date (e.g., 2025-08-30) |
| description | string | yes | free text |
| counterparty\_id | string | no | optional (customer/vendor id) |
| amount | float | yes | +inflow / –outflow |
| running\_balance | float | no | if present, last row (by date) is balance |

**Balance rule:**  
If running\_balance exists → current\_balance = last(running\_balance ordered by date)  
Else → current\_balance = sum(amount)

**4.2 ar\_invoices.csv**

| **column** | **type** | **required** | **notes** |
| --- | --- | --- | --- |
| invoice\_id | string | yes | unique |
| customer\_id | string | yes | CUST# |
| invoice\_date | ISO date | yes |  |
| due\_date | ISO date | yes |  |
| amount | float | yes |  |
| status | enum | yes | open | paid |
| paid\_date | ISO date | no | required if status == "paid" |

**4.3 ap\_bills.csv**

| **column** | **type** | **required** | **notes** |
| --- | --- | --- | --- |
| bill\_id | string | yes | unique |
| vendor\_id | string | yes | VEND# |
| vendor\_tier | enum | yes | critical | regular |
| bill\_date | ISO date | yes |  |
| due\_date | ISO date | yes |  |
| amount | float | yes |  |
| status | enum | yes | open | paid |
| paid\_date | ISO date | no | required if status == "paid" |

**4.4 policy.json**

{

"currency": "INR",

"min\_operating\_cash": 1000000,

"payroll\_buffer": 400000,

"tax\_buffer": 200000,

"vendor\_tier\_buffers": { "critical": 300000, "regular": 100000 },

"approval\_threshold": 500000,

"whitelist": [

{ "instrument": "Liquid\_Fund\_Overnight", "issuer": "ABC AMC", "max\_amount": 5000000, "max\_tenor\_days": 1 }

],

"recognition\_ratio\_expected\_inflows": 0.40,

"outflow\_shock\_multiplier": 1.15,

"enforce\_cutoff": true,

"cutoff\_hour\_ist": 14

}

**4.5 cutoff\_calendar.json**

{

"timezone": "Asia/Kolkata",

"business\_days": ["Mon","Tue","Wed","Thu","Fri"],

"cutoff\_hour\_ist": 14,

"market\_cutoffs": {

"Liquid\_Fund\_Overnight": { "order\_cutoff\_hour\_ist": 14, "settlement\_t\_plus\_days": 0 }

},

"holidays\_2025": ["2025-01-26","2025-08-15","2025-10-02"]

}

**5) Reason Codes (Enumerated)**

REASONS = {

* FIXED\_BUFFERS: applied operating + payroll + tax + vendor tier buffers
* OUTFLOW\_SHOCK: applied outflow shock multiplier to horizon outflows
* CONSERVATIVE\_INFLOW: recognized only a fraction of expected inflows pre-settlement
* WL\_OK: instrument/issuer within whitelist & caps
* CUTOFF\_PASSED: suppressed order due to market cutoff
* MAKER\_CHECKER: amount ≥ approval threshold
* NO\_SURPLUS: deployable ≤ 0  
  }

**6) Module Contracts (Functions & Behavior)**

**6.1 config.py**

from dataclasses import dataclass

from pathlib import Path

from typing import Any, Dict

@dataclass

class Settings:

data\_dir: Path

policy: Dict[str, Any]

calendar: Dict[str, Any]

def load\_settings(data\_dir: str | Path) -> Settings:

"""

Reads policy.json and cutoff\_calendar.json from data\_dir.

Returns structured Settings.

"""

**6.2 parse.py**

import pandas as pd

from pathlib import Path

def load\_bank(p: Path) -> pd.DataFrame:

"""Required cols: date, description, amount. Optional: counterparty\_id, running\_balance."""

def load\_ar(p: Path) -> pd.DataFrame:

"""Required cols as per AR schema; parse invoice\_date, due\_date."""

def load\_ap(p: Path) -> pd.DataFrame:

"""Required cols as per AP schema; parse bill\_date, due\_date."""

def current\_balance(bank\_df: pd.DataFrame) -> float:

"""

If 'running\_balance' exists → last value after sorting by date ascending,

else sum of 'amount'.

"""

**Validation:** If required columns are missing → raise ValueError('missing columns: ...').

**6.3 predict.py**

**Heuristic rules**

* AR collection probability by days\_to\_due:
  + < 0: 0.85
  + <= 7: 0.70
  + <=14: 0.50
  + else: 0.30
* AP within horizon: 1.0 (certain).

import pandas as pd

from datetime import date

def invoice\_pay\_prob(days\_to\_due: int, status: str) -> float: ...

def horizon\_flows(

ar\_df: pd.DataFrame,

ap\_df: pd.DataFrame,

horizon\_days: int = 7

) -> tuple[float, float, pd.DataFrame, pd.DataFrame]:

"""

Returns (expected\_inflows, expected\_outflows, ar\_h\_df, ap\_h\_df) for invoices/bills with due\_date in [today, today+horizon].

ar\_h\_df includes columns: invoice\_id, due\_date, amount, prob

ap\_h\_df includes columns: bill\_id, due\_date, amount, vendor\_tier

"""

**6.4 prescribe.py**

**Policy math**

* must\_keep = min\_operating\_cash + payroll\_buffer + tax\_buffer + Σ vendor\_tier\_buffer(for AP in horizon) + outflow\_shock\_multiplier \* expected\_outflows
* deployable = max(0, balance + recognition\_ratio\_expected\_inflows \* expected\_inflows - must\_keep)

**Allocation rules**

* If enforce\_cutoff and local time hour ≥ cutoff\_hour\_ist → **no order** (reason: CUTOFF\_PASSED).
* Else choose first whitelist instrument; proposed\_amount = min(deployable, whitelist.max\_amount).
* needs\_maker\_checker = (proposed\_amount >= approval\_threshold).

from typing import List, Dict, Tuple

def must\_keep(policy: Dict, expected\_outflows: float, ap\_rows: list[Dict]) -> float: ...

def deployable(balance: float, expected\_inflows: float, must\_keep\_amt: float, policy: Dict) -> float:

"""Use policy['recognition\_ratio\_expected\_inflows']; clamp ≥ 0; round to 2dp."""

def propose\_order(deployable\_amt: float, policy: Dict) -> tuple[Dict | None, List[str]]:

"""

Returns (order\_dict\_or\_none, reason\_codes)

order\_dict: { "proposed": float, "instrument": str, "issuer": str, "needs\_maker\_checker": bool }

"""

**6.5 perform.py**

import json, uuid

from datetime import datetime

from pathlib import Path

def submit\_order\_stub(order: dict, out\_dir: str | Path) -> dict:

"""

Simulates order lifecycle (submitted -> placed -> settled) and writes to outputs/order\_{uuid}.json.

Returns final state dict.

"""

**6.6 present.py**

from datetime import date

from typing import List, Dict

def eod\_markdown(

balance: float, inflows: float, outflows: float, must\_keep: float, deployable: float,

order: Dict | None, reasons: List[str], horizon\_days: int

) -> str:

"""

Returns Markdown string with KPIs, Proposed Action, Reason Codes.

"""

**7) CLI Orchestration (src/cli.py)**

**Arguments**

* --data-dir (default data)
* --out-dir (default outputs)
* --horizon (int, default 7, range 3..14)
* --execute (flag; if present, perform simulated execution when order exists)

**Flow (pseudocode)**

settings = load\_settings(args.data\_dir)

bank = load\_bank(settings.data\_dir)

ar = load\_ar(settings.data\_dir)

ap = load\_ap(settings.data\_dir)

bal = current\_balance(bank)

inflows, outflows, ar\_h, ap\_h = horizon\_flows(ar, ap, args.horizon)

mk = must\_keep(settings.policy, outflows, ap\_h.to\_dict(orient="records"))

dep = deployable(bal, inflows, mk, settings.policy)

order, reasons = propose\_order(dep, settings.policy)

summary = {

"balance": bal,

"expected\_inflows": inflows,

"expected\_outflows": outflows,

"must\_keep": mk,

"deployable": dep,

"order": order,

"reasons": reasons,

"horizon\_days": args.horizon

}

write outputs/summary.json

write outputs/EOD\_Summary.md via eod\_markdown(...)

if args.execute and order and order["proposed"] > 0:

state = submit\_order\_stub(order, args.out\_dir)

summary["execution\_state"] = state

rewrite outputs/summary.json with state included

print("Wrote outputs to", args.out\_dir)

**summary.json example**

{

"balance": 2615000.0,

"expected\_inflows": 820000.0,

"expected\_outflows": 610000.0,

"must\_keep": 2015000.0,

"deployable": 185000.0,

"order": {

"proposed": 185000.0,

"instrument": "Liquid\_Fund\_Overnight",

"issuer": "ABC AMC",

"needs\_maker\_checker": false

},

"reasons": ["FIXED\_BUFFERS","OUTFLOW\_SHOCK","CONSERVATIVE\_INFLOW","WL\_OK"],

"horizon\_days": 7

}

**8) Determinism, Rounding, Time**

* **Rounding:** Monetary outputs (must\_keep, deployable, order.proposed) → round to **2 decimals**.
* **Determinism:** No randomness. Computation must depend only on file inputs and current local time (for cutoff).
* **Cutoff time:** Use local wall-clock hour; treat cutoff\_hour\_ist as a simple hour threshold (sufficient for demo).

**9) Error Handling & Validation**

* **Missing columns** → ValueError("missing columns: ...").
* **Non-numeric amounts** → coerce to float or error.
* **Invalid enums** (status, vendor\_tier) → ValueError.
* **Negative deployable** → clamp to 0.0; add NO\_SURPLUS.
* **Cutoff suppression** → return order=None (or proposed=0), add CUTOFF\_PASSED.

**10) Tests (tests/test\_core\_loop.py)**

**E2E Smoke**

from core.config import load\_settings

from core.parse import load\_bank, load\_ar, load\_ap, current\_balance

from core.predict import horizon\_flows

from core.prescribe import must\_keep, deployable, propose\_order

def test\_end\_to\_end():

settings = load\_settings("data")

bank = load\_bank(settings.data\_dir)

ar = load\_ar(settings.data\_dir)

ap = load\_ap(settings.data\_dir)

bal = current\_balance(bank)

inflows, outflows, ar\_h, ap\_h = horizon\_flows(ar, ap, 7)

mk = must\_keep(settings.policy, outflows, ap\_h.to\_dict(orient="records"))

dep = deployable(bal, inflows, mk, settings.policy)

order, reasons = propose\_order(dep, settings.policy)

assert bal >= 0 and mk > 0 and dep >= 0

assert isinstance(reasons, list)

**11) Optional Streamlit UI (single file, fast demo)**

* Show 5 KPIs (Balance, Inflows, Outflows, MustKeep, Deployable)
* Proposed order card (+ maker–checker badge)
* Expanders with AR/AP rows for the horizon
* Buttons: “Generate EOD Summary”, “Simulate Execution”

*(UI is optional; CLI is the authoritative artifact.)*

**12) Implementation Order (Checklist)**

1. reason\_codes.py (constants)
2. config.py (load JSONs)
3. parse.py (loaders + current\_balance)
4. predict.py (invoice\_pay\_prob, horizon\_flows)
5. prescribe.py (must\_keep, deployable, propose\_order)
6. present.py (eod\_markdown)
7. perform.py (submit\_order\_stub)
8. cli.py (wire it together)
9. tests/test\_core\_loop.py (E2E smoke)

**Acceptance:** Running the CLI produces outputs/summary.json and outputs/EOD\_Summary.md reflecting inputs, with correct reason codes and cutoff behavior.

**13) Notes for Implementation**

* Keep functions pure where possible; pass in policy parameters; avoid global state.
* Keep file I/O isolated (CLI and config.py); keep compute modules pure and testable.
* Use only standard data types in JSON artifacts (no numpy types).

This spec is designed so Claude can implement the modules, function signatures, CLI, and tests *directly* without further interpretation. If you want, I can also provide a minimal requirements.txt, pyproject.toml, and empty file stubs with TODOs to accelerate coding.