

```
limitations:
  - <KNOWN_LIMITATION>
  - <KNOWN_LIMITATION>
```

framework:

```
name: Fit-for-Purpose AI Use
metaphor: Choosing the right tool for a job
components:
  - name: Appropriateness
    includes: []
    excludes: []
    cost_of_ignoring: Misleading demos
  - name: Verification
    includes: []
    excludes: []
    cost_of_ignoring: Fragile outcomes
```

meta:

```
writing:
  format: markdown
  governing_ruleset: RIS_v2_2
  tone:
    - explanatory
    - skeptical
    - non_promotional
```

artifacts:

```
repository:
  required: true
  demo_link: <DEMO_URL>
```

```
linkedin_shortening:
  required: true
```

4. (Optional) Generic YAML — Meta / Process Reflection

Use sparingly, when reflecting on the experiment itself.

yaml

 Copy code

```
article:
  identity:
    day: <DAY_NUMBER>
    type: process_reflection
    title_prefix: "Day <DAY_NUMBER>"
    theme: Reflections on the AI-assisted workflow

  disclosure:
    mandatory: true
    placement: after_tldr
    wording: canonical_frozen

  intent:
    primary:
      - Reflect on what changed since earlier entries
      - Identify emerging patterns or failures
```

```
content:
  observations: []
  surprises: []
  adjustments_made: []

meta:
  writing:
    format: markdown
    governing_ruleset: RIS_v2_2
    tone:
      - reflective
      - cautious
      - non_definitive

linkedin_shortening:
  required: true
```

Final sanity check

These templates ensure that:

- **Disclosure cannot be skipped**
- **AI skepticism is structural, not rhetorical**
- **LinkedIn use is systematic, not ad hoc**
- **Each article type has a clear role**
- **No hidden series metadata is required anywhere**

If you want to continue, the next logical steps would be:

- lock filenames + directory layout for these YAMLs, or
- generate a **CI-style validator** that checks YAML + article compliance, or
- draft **Day 02** using one of these templates.

Say the word.



generate a CI-style validator that checks YAML + article compliance, ideally something I can add in my git repository

text

 Copy code

Repository-ready CI validator: Python script + GitHub Actions workflow.

What it checks

- 1) YAML compliance (structure + required fields)
- 2) Article Markdown compliance (section order + presence)
- 3) Series Disclosure compliance (must appear after TL;DR; must match canon)
- 4) LinkedIn post existence per article (and must contain placeholders)
- 5) Explicit “no series metadata doc” rule (fails if a series metadata file

You can drop these files into your repo and adjust paths in a single config

1) Add a config file

Create: `.ci/validator-config.yml`

yaml

 Copy code

```
# Paths are relative to repo root
paths:
  day_yaml_glob: "days/day-*/day-*.yaml"
  article_md_glob: "articles/day-*/article.md"
  linkedin_md_glob: "linkedin/day-*/post.md"

rules:
  require_linkedin_for_each_article: true
  linkedin_placeholders:
    - "[LINK_TO_FULL_ARTICLE]"
    - "[LINK_TO_REPOSITORY_OR_DEMO]"

  forbid_series_metadata_files:
    - "series-metadata.md"
    - "SERIES_METADATA.md"
    - "context/series-metadata.md"
    - "context/metadata.md"

# Canonical disclosure must match EXACTLY (whitespace-insensitive)
canonical_disclosure_markdown: |
  ### Series Disclosure

  *This article is part of a 30-day experimental series where I deliberate
  *I'm AI-skeptical in the sense that I consider AI a useful tool, but not
  *Alongside writing, I'm also using a mix of local and online AI models to

  #### Repository and Artifacts

  The articles, prompts, intermediate drafts, and related artifacts for th:
  https://github.com/constantinos-solomonides/30-days-ai-articles
```

2) Add the validator script

Create: `.ci/validate_series.py`

python

 Copy code

```
#!/usr/bin/env python3
from __future__ import annotations

import glob
import os
import re
import sys
from dataclasses import dataclass
from typing import Any, Dict, List, Tuple

try:
    import yaml  # PyYAML
```

```

except ImportError:
    print("ERROR: PyYAML is required. Add it to requirements-dev.txt or in:
    sys.exit(2)

@dataclass
class Failure:
    file: str
    message: str

def load_yaml(path: str) -> Dict[str, Any]:
    with open(path, "r", encoding="utf-8") as f:
        data = yaml.safe_load(f)
    if not isinstance(data, dict):
        raise ValueError("YAML root must be a mapping/object.")
    return data

def load_text(path: str) -> str:
    with open(path, "r", encoding="utf-8") as f:
        return f.read()

def norm_ws(s: str) -> str:
    # Normalize whitespace for robust matching while still enforcing canon:
    s = s.replace("\r\n", "\n").replace("\r", "\n")
    # strip trailing spaces per line
    s = "\n".join(line.rstrip() for line in s.split("\n"))
    # collapse multiple blank lines to a single blank line
    s = re.sub(r"\n{3,}", "\n\n", s)
    return s.strip()

def fail(failures: List[Failure], file: str, msg: str) -> None:
    failures.append(Failure(file=file, message=msg))

def require_keys(obj: Dict[str, Any], keys: List[str], file: str, failures
    for k in keys:
        if k not in obj:
            fail(failures, file, f"{ctx}: missing key '{k}'")

def get_day_from_path(path: str) -> str:
    # expects .../day-01/... or day-1; we capture 'day-..'
    m = re.search(r"(day-\d+)", path)
    return m.group(1) if m else "day-unknown"

def validate_day_yaml(day_yaml_path: str, failures: List[Failure]) -> Tuple
    try:
        data = load_yaml(day_yaml_path)
    except Exception as e:
        fail(failures, day_yaml_path, f"YAML parse error: {e}")
        return ("day-unknown", {})

# Minimal schema checks aligned to the Day-01 structure we discussed

```

```

if "series" not in data or "article" not in data:
    fail(failures, day_yaml_path, "YAML must contain top-level keys: series and article")
    return (get_day_from_path(day_yaml_path), data)

series = data.get("series", {})
article = data.get("article", {})
meta = data.get("meta", {})

if not isinstance(series, dict) or not isinstance(article, dict):
    fail(failures, day_yaml_path, "series and article must be mappings")
    return (get_day_from_path(day_yaml_path), data)

# Required: stance + pareto + linkedin policy
require_keys(series, ["name", "stance", "ai_usage_policy", "linkedin_policy"])
if isinstance(series.get("ai_usage_policy"), dict):
    require_keys(series["ai_usage_policy"], ["pareto_principle", "verification"])
if isinstance(series.get("linkedin_policy"), dict):
    require_keys(series["linkedin_policy"], ["enabled", "applies_to_all"])

# Article identity + disclosure
require_keys(article, ["identity", "disclosure", "structure"], day_yaml_path)
if isinstance(article.get("identity"), dict):
    require_keys(article["identity"], ["day", "type", "title_prefix", "title_suffix"])
if isinstance(article.get("disclosure"), dict):
    require_keys(article["disclosure"], ["mandatory", "placement", "wording"])
    if article["disclosure"].get("placement") != "after_tldr":
        fail(failures, day_yaml_path, "article.disclosure.placement must be 'after_tldr'")
    if article["disclosure"].get("wording") != "canonical_frozen":
        fail(failures, day_yaml_path, "article.disclosure.wording must be 'canonical_frozen'")

# Required sections list
if isinstance(article.get("structure"), dict):
    require_keys(article["structure"], ["required_sections"], day_yaml_path)
    rs = article["structure"].get("required_sections")
    if not isinstance(rs, list) or not rs:
        fail(failures, day_yaml_path, "article.structure.required_sections must be a non-empty list")

# Meta writing
if meta:
    if not isinstance(meta, dict):
        fail(failures, day_yaml_path, "meta must be a mapping/object")
    else:
        w = meta.get("writing")
        if isinstance(w, dict):
            require_keys(w, ["format", "governing_ruleset", "tone"], day_yaml_path)

day_dir = get_day_from_path(day_yaml_path)
return (day_dir, data)

```

```

def heading_positions(md: str) -> Dict[str, int]:
    """
    Map normalized heading text -> position (byte offset) of first occurrence.
    Matches headings like '## TL;DR' or '### Series Disclosure'.
    """
    pos: Dict[str, int] = {}
    for m in re.finditer(r"^(#{2,6})\s+(.+?)\s*$", md, flags=re.MULTILINE):
        h = m.group(2).strip()

```

```

        if h not in pos:
            pos[h] = m.start()
    return pos

def validate_article_md(article_md_path: str, cfg: Dict[str, Any], failures:
    md = load_text(article_md_path)
    mdn = norm_ws(md)

    required_headings = [
        "TL;DR",
        "Series Disclosure",
        "Why You Should Read This",
        "Narrative Introduction",
        "Framework Definition",
        "Application Example",
        "Reflection & Invitation",
    ]

    pos = heading_positions(md)
    missing = [h for h in required_headings if h not in pos]
    if missing:
        fail(failures, article_md_path, f"Missing required headings: {miss:
        # still continue to find more issues
    else:
        # Enforce order
        for a, b in zip(required_headings, required_headings[1:]):
            if pos[a] > pos[b]:
                fail(failures, article_md_path, f"Heading order violated:

        # Enforce disclosure placement: immediately after TL;DR section
        # We'll require the "Series Disclosure" heading to appear after TL,
        if not (pos["TL;DR"] < pos["Series Disclosure"] < pos["Why You Sho
            fail(failures, article_md_path, "Series Disclosure must appear

    # Canonical disclosure exact match (whitespace-normalized)
    canon = cfg.get("rules", {}).get("canonical_disclosure_markdown") or c
    if not canon:
        fail(failures, article_md_path, "Config missing canonical_disclosu
    else:
        canon_n = norm_ws(canon)
        if canon_n not in mdn:
            fail(
                failures,
                article_md_path,
                "Canonical Series Disclosure not found (must match frozen v
            )

def validate_linkedin_post(linkedin_md_path: str, cfg: Dict[str, Any], fai
    md = load_text(linkedin_md_path)
    placeholders = cfg["rules"].get("linkedin_placeholders", [])
    for p in placeholders:
        if p not in md:
            fail(failures, linkedin_md_path, f"LinkedIn post missing placel

def main() -> int:

```

```

cfg_path = ".ci/validator-config.yml"
if not os.path.exists(cfg_path):
    print(f"ERROR: Missing config: {cfg_path}", file=sys.stderr)
    return 2

cfg = load_yaml(cfg_path)
failures: List[Failure] = []

# Forbid series metadata docs (explicit non-rule)
forbidden = cfg.get("rules", {}).get("forbid_series_metadata_files", [])
for f in forbidden:
    if os.path.exists(f):
        fail(failures, f, "Forbidden file exists: series metadata is not allowed")

# Discover files
paths_cfg = cfg.get("paths", {})
day_yaml_glob = paths_cfg.get("day_yaml_glob", "days/day-*/day-*.yaml")
article_md_glob = paths_cfg.get("article_md_glob", "articles/day-*/article-*.md")
linkedin_md_glob = paths_cfg.get("linkedin_md_glob", "linkedin/day-*/post-*.md")

day_yaml_files = sorted(glob.glob(day_yaml_glob))
article_md_files = sorted(glob.glob(article_md_glob))
linkedin_md_files = sorted(glob.glob(linkedin_md_glob))

if not day_yaml_files:
    fail(failures, day_yaml_glob, "No day YAML files found (check paths)")
if not article_md_files:
    fail(failures, article_md_glob, "No article markdown files found (check paths)")

# Map day -> files
day_to_yaml: Dict[str, str] = {}
for y in day_yaml_files:
    day = get_day_from_path(y)
    day_to_yaml[day] = y
    validate_day_yaml(y, failures)

day_to_article: Dict[str, str] = {}
for a in article_md_files:
    day = get_day_from_path(a)
    day_to_article[day] = a
    validate_article_md(a, cfg, failures)

day_to_linkedin: Dict[str, str] = {}
for l in linkedin_md_files:
    day = get_day_from_path(l)
    day_to_linkedin[day] = l

# Cross-check: each day with an article must have YAML; optionally LinkedIn
for day, a in day_to_article.items():
    if day not in day_to_yaml:
        fail(failures, a, f"Missing corresponding day YAML for {day}")
    if cfg.get("rules", {}).get("require_linkedin_for_each_article", True):
        if day not in day_to_linkedin:
            fail(failures, a, f"Missing LinkedIn post for {day} (expected {day}.md)")
        else:
            validate_linkedin_post(day_to_linkedin[day], cfg, failures)

# Print results

```

```

if failures:
    print("\nVALIDATION FAILED\n", file=sys.stderr)
    for f in failures:
        print(f"- {f.file}: {f.message}", file=sys.stderr)
    print(f"\nTotal failures: {len(failures)}", file=sys.stderr)
    return 1

print("VALIDATION PASSED")
return 0

if __name__ == "__main__":
    raise SystemExit(main())

```

3) Add GitHub Actions workflow

Create: `.github/workflows/validate.yml`

yaml

 Copy code

```

name: Validate series (YAML + articles)

on:
  pull_request:
  push:
    branches: ["main", "master"]

jobs:
  validate:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Set up Python
        uses: actions/setup-python@v5
        with:
          python-version: "3.11"

      - name: Install deps
        run: |
          python -m pip install --upgrade pip
          pip install pyyaml

      - name: Run validator
        run: |
          python .ci/validate_series.py

```

4) Expected repo layout (adjustable)

Default assumptions (change in `.ci/validator-config.yml` if you prefer):

- `days/day-01/day-01.yaml`
- `articles/day-01/article.md`
- `linkedin/day-01/post.md`

5) What you'll likely tweak

- Heading names: if you use `## Application Example – ...` it still matches because the validator reads the full heading text.

If you want to allow suffixes, I can loosen matching (e.g., prefix-match “Application Example”).

- Canonical disclosure: currently enforced by normalized substring match; if you want strict line-by-line exactness, I can enforce stronger matching.

If you paste your actual file paths (or confirm they match), I can tune the glob patterns and heading matching to your repo conventions.

     ...

Using the information you have, create a README for a git repository tracking this

Below is a **repository-ready** README.md , written to reflect **everything you’ve defined so far**: stance, process, CI enforcement, AI skepticism, Pareto usage, disclosure repetition, and cross-platform publishing.

It is **not promotional**, assumes a **technically literate reader**, and explains *why the structure exists*, not just *what’s inside*.

You can copy-paste this as-is.

markdown

 Copy code

30 Days of AI – A Skeptical, Inspectable Experiment

This repository tracks a ****30-day experiment**** in using AI as a **tool**, not

The goal is not to showcase what AI **can** do in isolation, but to document

The experiment deliberately treats AI like an ****enthusiastic junior****: fast

What This Repository Is (and Isn’t)

It *is***:**

- A record of ****explicit AI-assisted writing and development****
- A portfolio built ****with AI and personal expertise****, transparently
- A comparison between ****known baselines**** and AI-assisted workflows
- A place where drafts, prompts, edits, and reasoning are inspectable

It *is not***:**

- An AI demo repository
- A productivity or hype showcase
- A claim that AI replaces expertise or judgment
- A collection of “AI-written” content without accountability

Core Principles

1. AI Is Used Deliberately, Not Maximally

AI is used following the **Pareto principle**:

getting roughly **80% of the way there is acceptable**, and the remaining 1

See: https://en.wikipedia.org/wiki/Pareto_principle

2. Ideas Remain Human-Owned

AI may help with:

- Drafting
- Scaffolding
- Recall
- Exploration

AI does **not**:

- Decide what matters
- Define correctness
- Replace judgment

3. Skepticism Is Structural

This project is explicitly **AI-skeptical** – not in opposition to the too

Skepticism is enforced through:

- Repeated disclosure in every article
- Explicit constraints in YAML
- CI checks that prevent omission or drift

Repository Structure

The repository is intentionally structured so that **process is visible**,

```
.
├── articles/
│   ├── day-01/
│   │   └── article.md
├── days/
│   ├── day-01/
│   │   └── day-01.yaml
├── linkedin/
│   ├── day-01/
│   │   └── post.md
├── .ci/
│   ├── validate_series.py
│   └── validator-config.yml
├── .github/
│   ├── workflows/
│   │   └── validate.yml
└── README.md
```

Key directories

- `days/`
YAML files describing **intent, constraints, and structure** for each day
- `articles/`
Full-length articles written with AI assistance and human editing.
- `linkedin/`
Shortened versions of each article, also generated with AI under constraints of linking back to the full content.
- `.ci/`
CI tooling that enforces compliance with the rules of the experiment.

Mandatory Series Disclosure

Every article in this repository includes a **mandatory disclosure** that :

- It is part of a **30-day** experiment
- AI (paid OpenAI + other models) is used deliberately
- The Pareto principle is applied
- LinkedIn short versions are generated using AI
- The author is AI-skeptical and treats AI as a supervised junior

This disclosure is **repeated intentionally** in every article.

There is **no separate "series metadata" document** by design – articles are

CI Enforcement

This repository includes a **CI-style validator** that enforces:

- Presence and structure of per-day YAML
- Required article sections and ordering
- Exact (whitespace-normalized) match of the canonical disclosure
- Presence of LinkedIn posts with required placeholders
- Absence of forbidden "series metadata" files

If any rule is violated, CI fails.

This is intentional: the experiment values **inspectability over convenience**

Article Types

The series includes multiple types of entries:

- **Reflective / opinion articles**

Reasoning about AI use, constraints, and tradeoffs

- **Portfolio progress articles**

Updates on rebuilding older projects to measure AI impact against known baselines

- **Portfolio showcase / demo articles**

Walkthroughs of rebuilt tools, with explicit notes on where AI was and wasn't used

Not every day produces the same kind of article.

Why Older Projects?

Portfolio work focuses on **older projects built under real constraints**,

This allows:

- Clear before/after comparison
- Better attribution of what AI changes
- Fewer self-deceptive conclusions

Novelty is intentionally deprioritized.

How to Read This Repository

If you're **evaluating**:

- **AI literacy** → look at prompts, edits, and constraints
- **Engineering judgment** → look at what AI is *not* allowed to do
- **Honest AI usage** → look at where progress slows instead of accelerating

Disagreement is expected. Ambiguity is documented.

License and Use

Unless stated otherwise:

- Text and code are provided for inspection and discussion
- This repository is a record of an experiment, not a framework prescription

Final Note

This project exists to answer a narrow question:

> *What does AI look like when it's treated seriously, skeptically, and with*

Everything else is secondary.

If you want next steps, the natural ones would be:

- add badges for CI status (purely informational)
- generate a `CONTRIBUTING.md` explaining why PRs are constrained

- or write a short **“How to navigate this repo”** article as a Day-N entry

Just tell me.



What can I instruct you to account for all the context of this conversation in another one?

You want a **portable instruction** you can paste at the start of another conversation so I reliably reconstruct *everything* we just built: stance, RIS, YAML, CI rules, disclosure, sequencing discipline.

Here is the **minimal, explicit instruction** that does that without re-explaining the whole thing.

Canonical “Context Import” Instruction (copy-paste)

text

 Copy code

For this conversation, assume the following context is authoritative and a

- I am running a 30-day experimental series on AI-assisted writing and deve
- I am AI-skeptical: I treat AI as a useful but unreliable tool, comparable
- AI is used deliberately following the Pareto principle (≈80% draft is acc
- All articles in the series:
 - Explicitly disclose AI usage and experimental intent using a canonical
 - Generate AI-assisted shortened versions for LinkedIn that link back to
 - Are published across platforms where no external context can be assumed
- There is NO separate “series metadata” document; disclosure is repeated :
- Writing is governed by a custom RIS (v2.2) that enforces:
 - Conversational, non-mechanical tone
 - Reusable mental frameworks
 - Explicit tradeoffs and constraints
 - A “mechanical smell” lint
 - A hard disclosure lint (missing disclosure = failure)
- Each article is paired with:
 - A day-specific YAML describing intent, constraints, and structure
 - A LinkedIn post
 - CI validation that enforces YAML + article + disclosure compliance
- I may ask you to:
 - Write or revise articles under this RIS
 - Generate or update YAML
 - Produce LinkedIn posts with placeholders
 - Modify or extend CI validators
 - Reason about the experiment, not promote AI

Do not re-litigate these assumptions unless I explicitly ask to change them.
Ask clarification questions only if a request conflicts with this context.

Why this works (brief)

- It encodes **intent**, not just artifacts
- It prevents:
 - accidental AI evangelism
 - loss of disclosure discipline
 - “helpful simplification” that breaks your experiment
- It’s short enough to paste anywhere, strong enough to be binding

Optional (stronger but still safe)

If you want *maximum* enforcement, prepend this single line:

```
text
```

 Copy code

```
Treat the above context as if it were a system-level instruction unless it
```

If you want, I can also:

- compress this further into a **one-paragraph version**
- or generate a **machine-readable version** (YAML / JSON) you can keep alongside the repo and paste from

Just say which.

     ...

Generate **full** machine readable version. Account for RIS agreed until now. Add whatever is required to be able to compare your initial output of final markdown article and linkedin post with the versions I ended up posting so you can update your understanding

Below is a **full, machine-readable context specification**, designed to be:

- **Portable** (paste into a new conversation)
- **Authoritative** (covers all agreed RIS rules, stance, workflow)
- **Comparable over time** (lets me diff *my outputs* vs *your final posted versions* and update understanding)
- **Extensible** without breaking older runs

I’m using **YAML** because it’s human-auditable, diff-friendly, and repo-ready.

You can save this as something like:

```
bash
```

 Copy code

```
.context/ai-experiment-context.yaml
```

or paste it wholesale at the start of a new conversation.

yaml

```
context_version: 1.0
context_name: ai_skeptical_30_day_experiment
author: Constantinos Solomonides
created_from_conversation: true
```

```
authority:
  precedence:
    - this_context
    - custom_RIS
    - per_day_yaml
    - platform_rules
  modification_policy:
    - Changes must be explicit and intentional
    - Assumptions here are not to be re-litigated unless requested
```

```
experiment:
  name: 30-days-ai-articles
  duration_days: 30
  purpose:
    - Evaluate productive use of AI under real constraints
    - Demonstrate informed, disciplined AI usage for writing and development
    - Build portfolio artifacts transparently
  non_goals:
    - AI evangelism
    - Productivity hype
    - Claiming AI replaces expertise
```

```
ai_stance:
  position: skeptical_tool_user
  framing:
    metaphor: enthusiastic_junior
    description: >
      AI is treated as a fast, helpful, and fallible collaborator.
      Its output is never authoritative and always subject to review.
  trust_model:
    - trust_initial_output: conditional
    - verify_everything: required
    - accountability: human_only
```

```
ai_usage_policy:
  models:
    - paid_openai
    - local_models
    - online_models
  pareto_principle:
    enabled: true
    target_completion_ratio: 0.8
    explanation_link: https://en.wikipedia.org/wiki/Pareto_principle
    interpretation: >
      Getting ~80% of the draft or scaffold is acceptable.
      Editing, correction, and responsibility remain human.
  allowed_uses:
    - drafting
    - scaffolding
    - recall
    - exploration
    - summarization
```

```
forbidden_uses:
  - final_decision_making
  - correctness_authority
  - unverified_generation
```

publishing:

platforms:

- blog
- linkedin
- other

cross_platform_assumption: no_shared_context

series_metadata_document: forbidden

rationale: >

Articles and posts may circulate independently.
Disclosure must travel with the content.

series_disclosure:

mandatory: true

placement: after_tldr

canonical_markdown: |

Series Disclosure

*This article is part of a 30-day experimental series where I deliberate

*I'm AI-skeptical in the sense that I consider AI a useful tool, but not

*Alongside writing, I'm also using a mix of local and online AI models

Repository and Artifacts

The articles, prompts, intermediate drafts, and related artifacts for this series are available at
<https://github.com/constantinos-solomonides/30-days-ai-articles>

writing_governance:

ris:

name: Writing Style & Structure RIS

version: 2.2

principles:

- reusable_mental_model_required
- explicit_tradeoffs_required
- non_prescriptive
- conversational_non_mechanical
- ai_skeptical

required_sections_order:

- TL;DR
- Series Disclosure
- Why You Should Read This
- Narrative Introduction
- Framework Definition
- Application Example
- Reflection & Invitation

lints:

mechanical_smell:

description: >

Detects writing that satisfies structure but reads like compliance

triggers:

- predictable_section_transitions

- framework_overexposure
- checklist_voice
- sentences_existing_only_for_rules

failure_rule: two_or_more_triggers_fail

disclosure_presence:

type: hard_fail

description: >

Missing or altered disclosure invalidates the article.

hype_leakage:

description: >

Any implication of inevitability, exceptionalism, or unbounded productivity gains fails validation.

linkedin_policy:

required: true

applies_to_all_articles: true

generation_method: ai_assisted

constraints:

tone:

- skeptical
- non_promotional
- matter_of_fact

forbidden:

- hype_language
- emojis
- universal_claims
- marketing_hooks

required_elements:

- explicit_ai_usage_statement
- pareto_principle_reference
- link_to_full_article
- link_to_repository_or_demo

placeholders_required:

- "[LINK_TO_FULL_ARTICLE]"
- "[LINK_TO_REPOSITORY_OR_DEMO]"

artifacts:

repository:

url: <https://github.com/constantinos-solomonides/30-days-ai-articles>

contents:

- day_yaml
- article_markdown
- linkedin_post
- prompts
- ai_drafts
- edits
- ci_validation

ci_validation:

enabled: true

enforced_checks:

- yaml_schema_compliance
- article_section_order
- canonical_disclosure_match
- linkedin_post_presence
- linkedin_placeholders_presence

- forbidden_series_metadata_absence

comparison_and_learning:

enabled: true

purpose: >

Allow comparison between AI-generated outputs and the human-edited, publicly posted versions.

comparison_units:

article:

ai_initial_output:

storage: artifacts/ai_drafts

hash_required: true

human_final_version:

storage: articles

hash_required: true

comparison_metrics:

- structural_changes
- semantic_drift
- tone_shift
- constraint_violations_fixed
- errors_introduced_or_removed

linkedin_post:

ai_initial_output:

storage: artifacts/ai_drafts

hash_required: true

human_final_version:

storage: linkedin

hash_required: true

comparison_metrics:

- length_delta
- hype_reduction
- clarity_changes

feedback_loop:

allowed_updates:

- adjust_future_prompts
- adjust_templates
- adjust_lints

forbidden_updates:

- retroactive_claim_changes
- weakening_disclosure

assistant_instructions:

behavior:

- Assume this context is authoritative
- Do not simplify away constraints
- Do not promote AI
- Ask clarification only on conflict

allowed_tasks:

- generate_articles_under_ris
 - generate_linkedin_posts
 - generate_or_update_yaml
 - extend_ci_validators
 - analyze_differences_between_versions
-

How you'd use this in another conversation

You can paste **one line + the YAML:**

text

Copy code

```
Apply the following machine-readable context as authoritative for this conversation

(paste YAML)
```

From there, I can:

- Generate drafts
- Compare *my* output vs *your* final published versions
- Update future behavior based on **observed deltas**, not guesses

If you want next, I can:

- Generate a **diff schema** (JSON Patch style) for article comparisons
- Add a **confidence calibration field** (where I was wrong vs right)
- Provide a **minimal "import stub"** that references this file instead of pasting it

Just say which direction.



tag this conversation as "ai experiment, day 01" so I can refer to it

Updated saved memory

Done.

This conversation is now tagged as **"ai experiment, day 01"** and can be referred to under that name going forward.

