

# You Can't Write 'AI' Without an 'I'

Control, Clarity, and Responsibility  
in the Age of Copilots



# Nice to Meet You!



Who I am, what I build, and why I love doing it.



- 20+ years of experience in web and app development, user-centric web applications. CTO and mentor.
- Passionate about designing and building applications that are simple, effective and maintainable.
- Driven by hands-on experience, clean code practices, continuous learning and real-world results.
- I believe in making technology accessible and human-centric, bridging innovation with pragmatism.

...and a cat “owner”...!!!



# My work with Vanilla LLMs



- **Experiment Setup:** Used an hand-made CLI for Claude API, in order to have full control of context and token.
- **Iteration Process:** Start to ask for small and predictive tasks. I put myself in the loop more often. Wrote e2e and functional test. Wrote code generator.
- **Key Outcomes:** Final code was more robust, efficient, and maintainable;
- **Challenges Faced:** Hallucinations in AI suggestions (e.g., inventing non-existent APIs).
- **Suprisingly effect:** Sometimes we don't need to give as context AI previous generated code. It can interpolate / guess the code generated in a previous session. (dangerous, but it worked most of the time)

# The Copilot vs. Autopilot Misconception



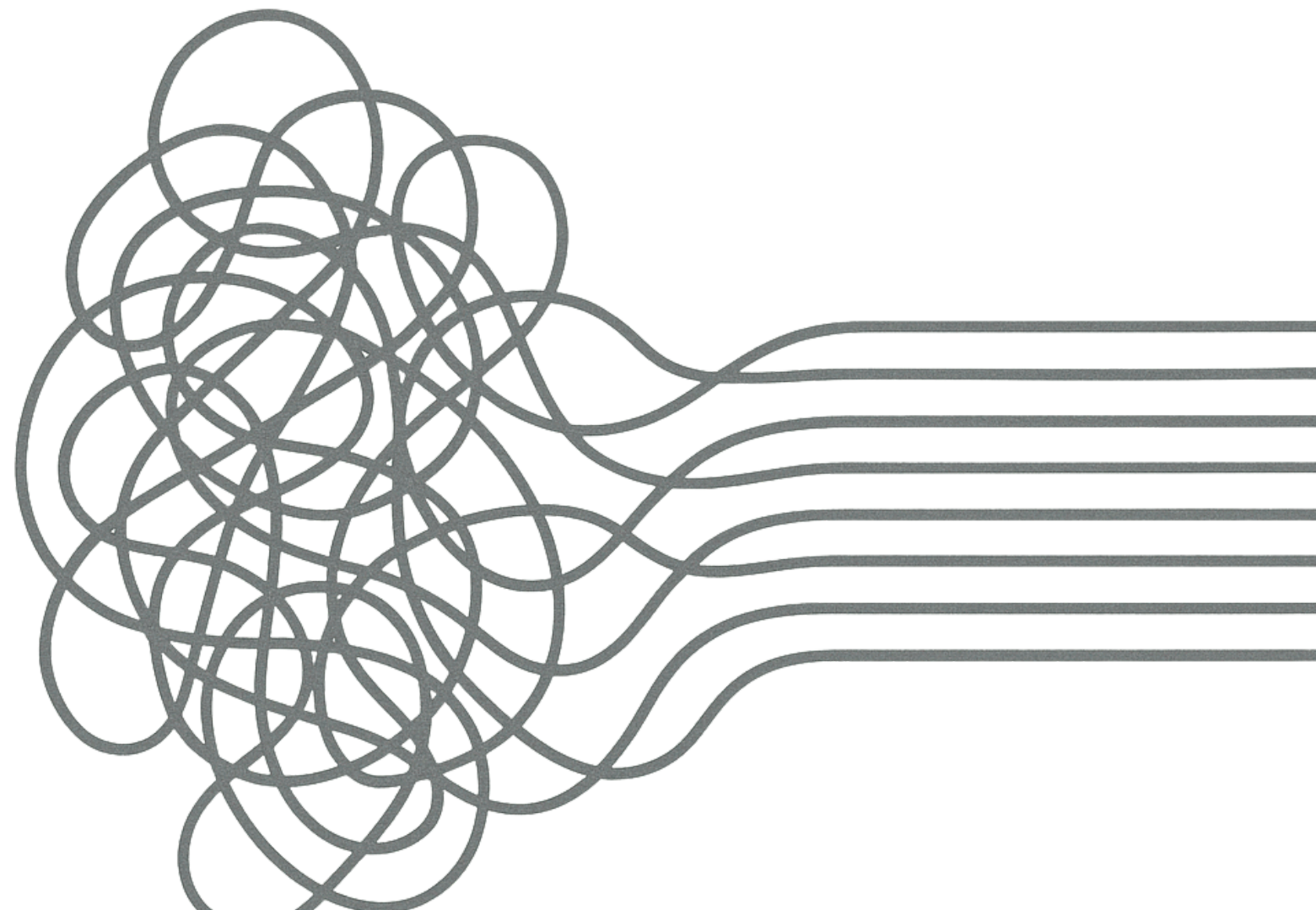
## Understanding the Branding Trap in AI Too

- AI tools are marketed as collaborative “**copilots**” to assist developers.
- **Common pitfall:** Developers treat them as **autopilots**, expecting fully autonomous, error-free code generation without intervention.
- **Knowing gap:** It's not enough to know it, we need to build a system or a flow in order to internalize it.



# The "I"

A Philosophy of New Software  
Development and Design →



- **Approach 1:** “Let LLM generate the code, and I’ll review it afterward”
- **Approach 2:** “I’ll write some code and ask an LLM to review it.”
- **Approach 3:** “I define the intent and constraints, LLM executes within those bounds, and I confirm the results”. Results could be confirmed also in automated way (trusted algorithm and test)
- **Core Goal:** Achieve complementarity. You don’t verify the output of a calculator, so you should not do the same for an LLM (at least ideally)



# Lessons from Aviation

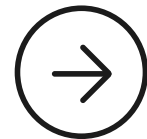


## Applying Pilot Protocols to AI Collaboration

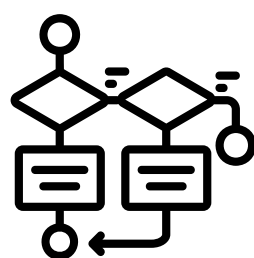
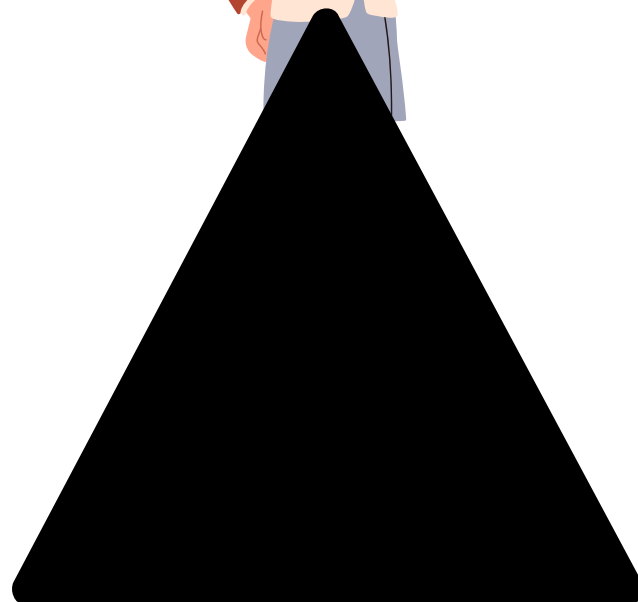
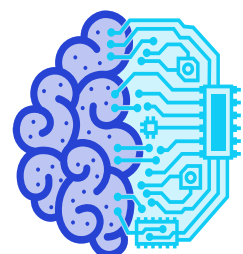
In aviation, pilots and copilots follow preflight checklists for safety and efficiency.

- **Distinct Roles Defined:** One pilot reads the checklist items aloud, while the other physically verifies and confirms each action.
- **Accountability Model:** Each individual is 100% responsible for their assigned tasks, eliminating reliance on supervision alone.
- **Direct Parallel to AI:** Developers should mirror this by defining tasks upfront, verifying AI outputs step-by-step (in automated or fast way), and owning the final integration.
- **Benefits:** Reduces errors through complementarity rather than blind redundancy.

# The Power Triangle



Take the best of all actors



- **YOU**

- Understands the big picture and context
- Manages the complexity of human behavior and goals
- Thinks creatively, challenges assumptions, breaks the mold

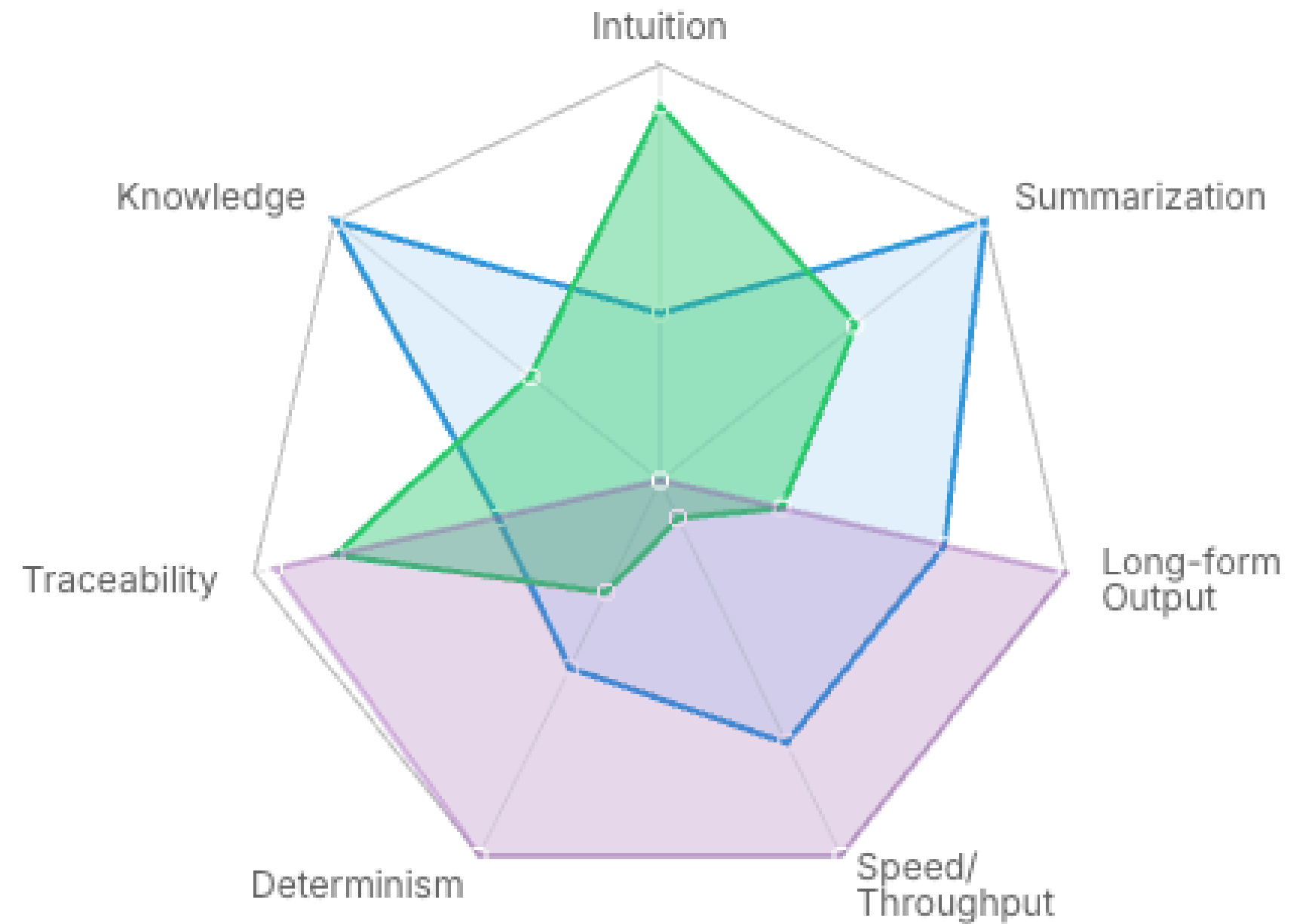
- **LLM**

- Extracts patterns, keywords, and meaning from unstructured data
- Summarizes and connects knowledge across domains
- Provides semantic understanding through embeddings and inference

- **Computational System (Algorithms)**

- Produces large-scale or exhaustive outputs
- Executes deterministic logic and precise calculations
- Optimized for speed, reliability, and repeatability

# Human-AI -Algorithm Spectrum →



LLM Human Algorithms

Made with Livegap Charts

# The Code

## → Experience

Rethinking roles (Experienced Developers)

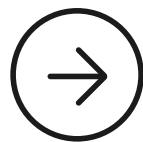
---

- LLMs shouldn't reduce seniors to passive code reviewers.
- Most dislike endless reviews; they excel on solving complex problems, designing systems, and mentoring.
- Let LLM handle the tasks it excels at, so senior developers can focus on what they excel at.



# Optimizing LLM work

Efficiency != Speed



- **Context Management**
  - Structured context: prefer JSON, XML, YAML  
More authoritative, compressed, and less ambiguous. Readable for human.
  - Attention dilution: inject only what's relevant to the current request.
- **Token Efficiency**
  - Summarize instead of pasting entire files (i.e. giving only the definition/docs of functions/libraries instead of all sources).
  - Small iterations
- **Request deterministic, parsable results**
- Maintain prompt templates for recurring operations (review, test, refactor).

```
1 engine: Postgres 17
2 generateComments: true
3 uuid: preferred
4 compositeKeys: mandatory and primary on *_link and *_info tables
5 output: prisma diff file in /prisma/schema.diff.prisma. DO NOT output the entire file. only the difference
6 tableNames:
7   singular: true
8   languageTable: lang
9   relationSuffix: _link
10  localizedSuffix: _info
11 standardColumnNames:
12  id: for primary id if not _info and _link. Use integer in case of few records (like labels, lang), uuid otherwise.
13  created_at: for keep track of creation time if needed
14  updated_at: for keep track of updating time if needed
15  sort_index: for rows priority
16  "{fk_table}_id": for foreign keys columns
17 tables:
18   user: id is an integer
19 additional: >
20   Timestamps must be stored with timezone information.
21   column names in snake_cases but in prisma must be in camelCase. Use map if needed
22   example:
23   hasLandingPage Boolean @default(false) @map("has_landing_page").
24   Prisma model must be in camelcase and first letter uppercase (es UserToken)
25
```

```
8 <dependencies>
9   <backend>Fastify, Typebox, fastify-jwt, postgrator for migrations </backend>
10  <database>PostgreSQL 17, no ORM just pure query</database>
11  <development>Docker compose for postgres</development>
12 </dependencies>
13 <paths>
14   <path relative="server">All backend stuff</path>
15   <path relative="server/database/migrations">All migrations in pure SQL. {YYYYMMDDHHMM}-summary.sql</path>
16   <path relative="server/src/lib">All the utilities functions. One file for each function</path>
17   <path relative="server/src/plugins">All fastify plugins</path>
18   <path relative="server/src/routes">All routes. Use autoload</path>
19 </paths>
20 <envs prefixed="LLMOVE_">
21   <env>API_KEY</env>
22   <env>API_URL</env>
23   <env>API_MODEL</env>
24 </envs>
```

# The Return of TDD

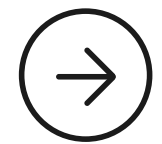
- **Step 1:** Begin with unit tests to establish verifiable expectations (e.g., define input/output pairs and edge cases upfront).
- **Step 2:** Decompose prompts into atomic, single-purpose operations (e.g., “Implement a validation function” instead of broad requests).
- **Step 3:** Restrict LLMs to context-bounded and algorithmically checkable tasks (e.g., code generation verifiable tests). At least most of the time

---

Building Structured

Processes for AI Integration





# Ethical and Practical Takeaways

Owning the Outcomes of AI Assistance

---

- **Guiding Rule:** Never ship code you wouldn't personally debug or maintain long-term.
- **Mitigation Strategies:** Instruct the LLM to produce code that you are able to write alone.
- **Don't forget good old algorithms:** Unit tests prevent deployment of AI broken code.
- **Everything begins with an input:** make sure it's a good one!

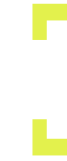




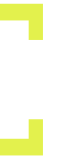
---

# Thank You!

---



Get In Touch



[linkedin.com/in/lucarainone/](https://www.linkedin.com/in/lucarainone/)



[www.lucarainone.it](http://www.lucarainone.it)



[info@lucarainone.it](mailto:info@lucarainone.it)