CopilotKit

Agentic Applications: 12 Do's and Don'ts

Lessons Learned on the Agentic Frontier

Swipe to next page

1. Agents Are Just Programs

Let's Diffuse Some Tensions:

MYTH

You hear the word agents a lot! It could sound bombastic, maybe scary 😱

VS

REALITY

Agents are just computer programs.

Nothing more.

That said, they do have some special properties:

- Use LLMs for reasoning/control flow.
- Long-running, async, interruptible and nondeterministic.
- Communicate via both structured & unstructured data.

In short: agents combine traditional program structure with LLM-powered reasoning and human-like communication.

Takeaways:

- **LLMs are pure functions:** tokens in → tokens out.
- **Output depends only on input** (prompt, context, tools).
- **Debugging trick:** put yourself in the agent's position.
- **Key takeaway:** treat agents like fast, slightly inattentive humans that only receive input (prompts, context, tools) and produce the output.

2. Don’t Wait for Perfection

It's really, really, really hard to get perfection out of a model or an agent.
Perfection means being able to ship the output (of the model or the agent) as-is without modification.

The Mirage of Perfection

You can get it to 70% in days... and 75% in weeks.

But ultimately approach an asymptote around 80% performance!

Design for Fault Tolerance

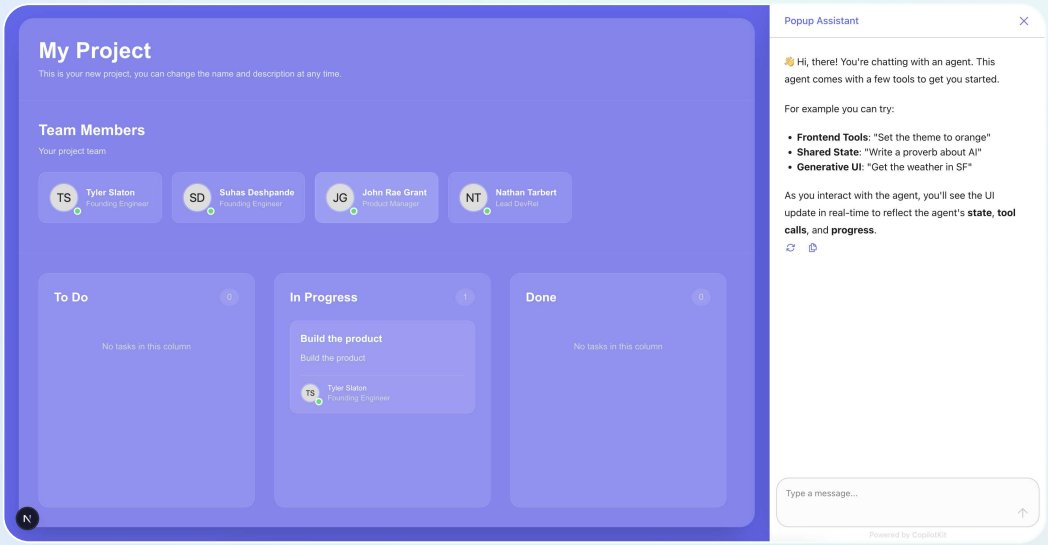
You have a building block that is only 80% reliable.

Ask yourself:
What can I do with it?

→ **It can be used where mistakes are either low-cost or easily fixable.**

Takeaways:

- Agents don’t need to be flawless to be useful.
- Real-world applications can provide value long before agents provide perfect autonomous reliability.
- **Example:** project management co-creation agent.



The agent and user excel at different things. Together they get more done, faster.

3. Pay Attention To Your Prompts

Prompts define how agents behave across situations

You can specify **what** to do, **how** to do it, the **approach** to take and the **context** to consider.

Because situations vary widely, prompts have huge influence. The same model can play wildly different roles - from doctor's notes to sales calls to code gen. Prompts are what shape its behavior in each case.

TIP

Models are surprisingly good at cleaning up messy prompts.

Takeaways:

- **Prompts shape behavior** in the same way code does.
- Specify **what, how, approach** and **context**.
- **Huge variability** → prompts matter.
- **Tip:** models are good at refining messy prompts.

4. Put Yourself in the Model's Place

LLMs Are Pure Functions: tokens in → tokens out

At each step, only the input matters - **prompt, context,** and **tools**.

If the agent isn't behaving as expected, **put yourself in its place:** could you act correctly with just that input? If not, the model won't either.

RULE OF THUMB

Human intuition is a strong heuristic for what an agent can (and can't) do.

Takeaways:

- **Don't treat LLMs as magic.** Put yourself in their shoes.
- Debug by checking **prompt, context,** and **available tools**.
- If it's **unclear to you**, it will be **unclear to the model**.

5. Context Is Everything

Agents can only act on the context they're given. If you ask for something without supplying the relevant information, the model won't have the grounding it needs.

Context can come from:

Build time expertise

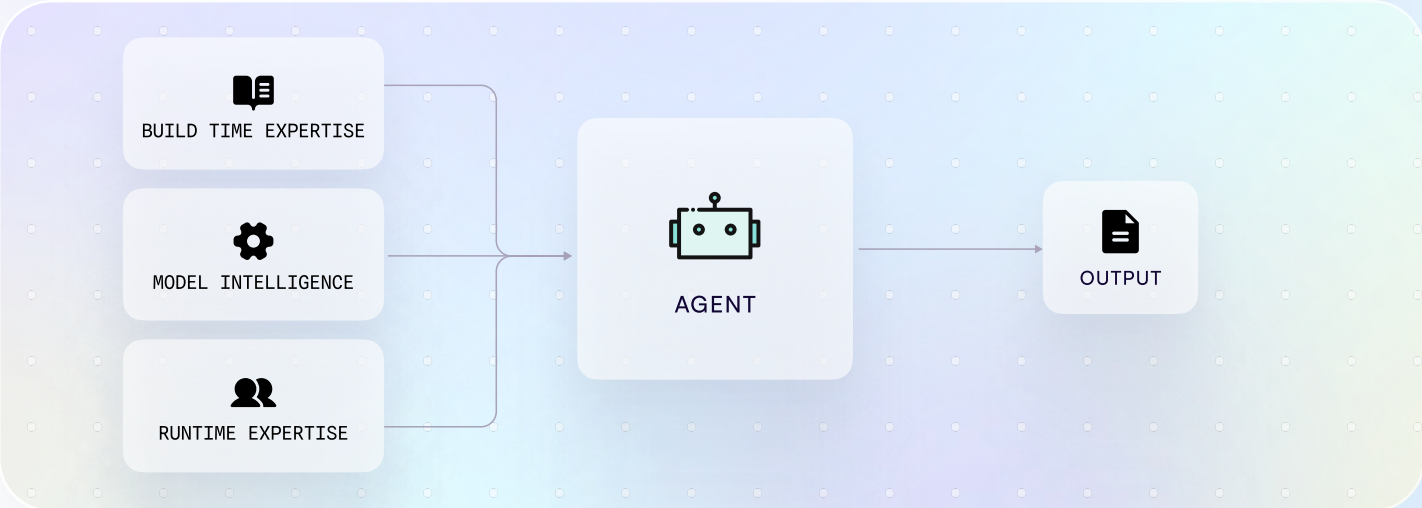
Domain expertise baked in by the developer.

Model intelligence

Model can intelligently gather context via tools.

Runtime expertise

Situational, task-specific knowledge



Takeaways:

- Never assume the model "just knows" → **provide context.**
- **Leverage all context** (build time expertise, model intelligence, users).
- The more **relevant context**, the more reliable the agent.

6. Leverage Your Users

Sources of Know-How and Context

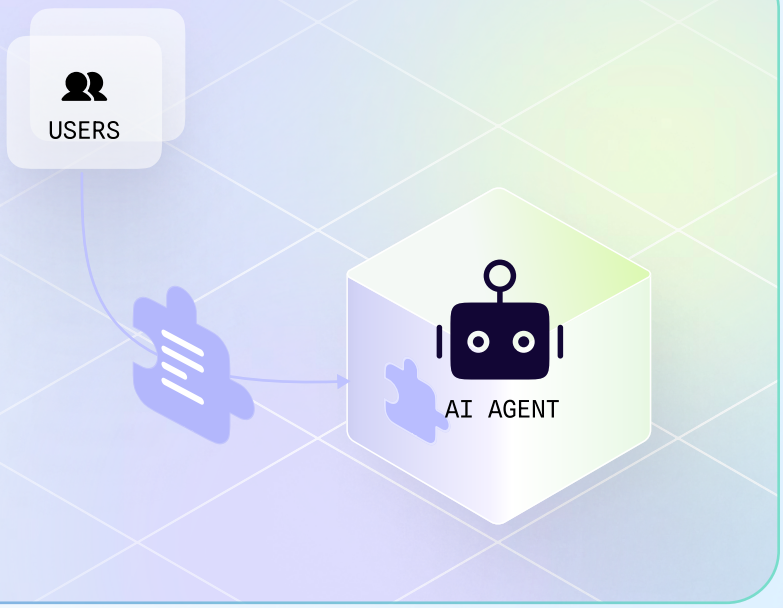
As mentioned, agents rely on both **know-how** (what to do) and **context** (the data they depend on).

We saw on the previous page that these can come from three places:

1. **Build time expertise**
2. **Model intelligence**
3. **Runtime expertise**

Context and know-how aren't just intelligence problems.

Even the smartest person would struggle in a new role if all they had was an onboarding wiki, because **each task is different.**



Takeaways:

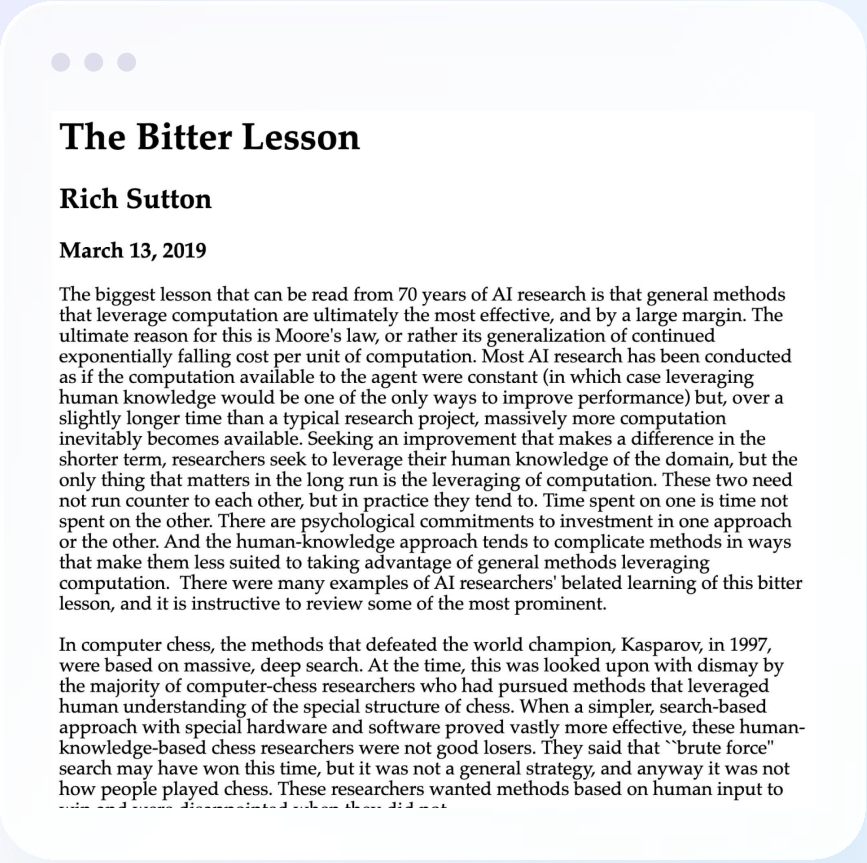
- Don’t rely solely on the model intelligence.
- Design agents so users can actively contribute their expertise at runtime.
- Recognize that user-supplied context is often the most important.

7. The Bitter Lesson

In machine learning, **the bitter lesson** is that simple techniques that scale with more data and compute outperform handcrafted approaches in the long run.

Applied to agentic applications, this means:

- Rely less on rigid, hard-coded logic.
- Give models more agency and let scale do the work.



The Bitter Lesson

Rich Sutton

March 13, 2019

The biggest lesson that can be read from 70 years of AI research is that general methods that leverage computation are ultimately the most effective, and by a large margin. The ultimate reason for this is Moore's law, or rather its generalization of continued exponentially falling cost per unit of computation. Most AI research has been conducted as if the computation available to the agent were constant (in which case leveraging human knowledge would be one of the only ways to improve performance) but, over a slightly longer time than a typical research project, massively more computation inevitably becomes available. Seeking an improvement that makes a difference in the shorter term, researchers seek to leverage their human knowledge of the domain, but the only thing that matters in the long run is the leveraging of computation. These two need not run counter to each other, but in practice they tend to. Time spent on one is time not spent on the other. There are psychological commitments to investment in one approach or the other. And the human-knowledge approach tends to complicate methods in ways that make them less suited to taking advantage of general methods leveraging computation. There were many examples of AI researchers' belated learning of this bitter lesson, and it is instructive to review some of the most prominent.

In computer chess, the methods that defeated the world champion, Kasparov, in 1997, were based on massive, deep search. At the time, this was looked upon with dismay by the majority of computer-chess researchers who had pursued methods that leveraged human understanding of the special structure of chess. When a simpler, search-based approach with special hardware and software proved vastly more effective, these human-knowledge-based chess researchers were not good losers. They said that ``brute force'' search may have won this time, but it was not a general strategy, and anyway it was not how people played chess. These researchers wanted methods based on human input to

Takeaways:

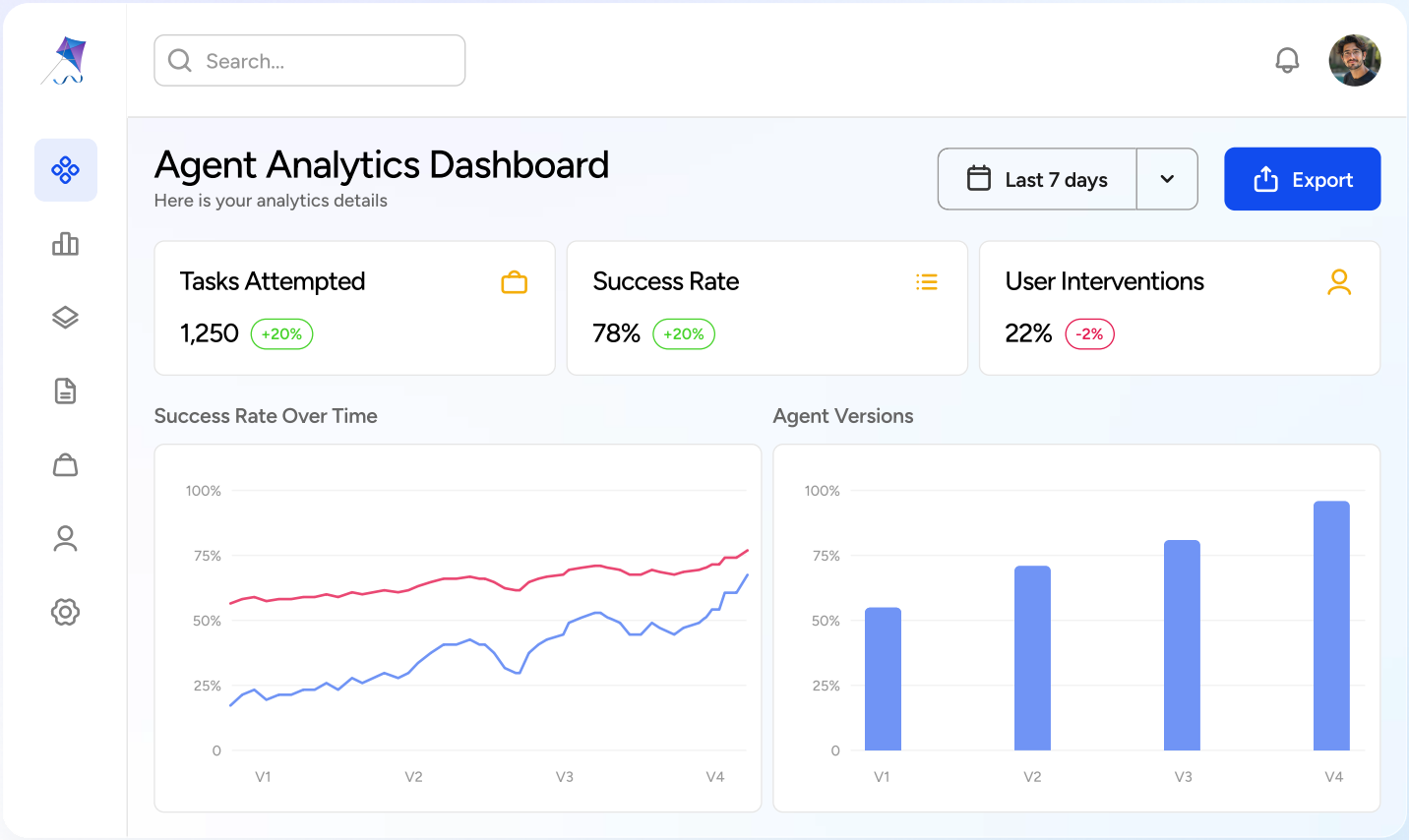
- Don’t over-engineer brittle flows - they won’t scale.
- Bet on methods that leverage compute and data growth.
- More agency often beats more rules.

8. Measure Everything

AI products are too complex to fully reason about. Instead of relying on intuition alone, let **data** guide iteration.

Large companies already use this mindset in product decisions. Now it needs to extend into the **functionality of agents themselves**.

Think of **evals** as probabilistic unit tests: they won't guarantee outcomes, but they help measure the impact of changes.



Takeaways:

- Don't rely on gut feel - use data to guide development.
- Treat evals like probabilistic testing.
- Always measure the effect of changes on real outcomes.
- AI systems too complex to fully reason about.

9. Push Toward On-the-Job Learning

Leaning on users for context is powerful, but you don't want them repeating the same corrections endlessly.

A major frustration with today's AI systems is their lack of **on-the-job learning**. Unlike humans, they don't improve with experience or training over time.

Takeaways:

- Don't force users to correct the same mistakes repeatedly.
- Aim for agents that adapt based on past interactions.
- Closing the gap with human "on-the-job learning" is key to long-term usefulness.

Ask us about AG-UI learning layer...

10. Intelligence Is a Vector, Not a Scalar

Consider the question:

How many **r's** are in the word **strawberry**? 🍓

HUMANS

- Use different parts of your intelligence to answer different questions.
- e.g. stream of consciousness, notes on scratch paper etc.
- Easy to see that the # of r's in "reverberatory" is hard to answer without a scratch pad.

VS

AI AGENTS

- LLM raw generation
- Use chain-of-thought to reach answers
- Reasoning + tools (e.g. write code)

Takeaways:

- Intelligence isn't one-dimensional.
- Design agents as systems, not single models.
- Real power comes from picking the right intelligence tools for the job.

11. UI Matters

The user interface defines how humans and agents collaborate

Chat

Open-ended communication between intelligent entities.

Generative UI

Chat that goes beyond text.

Canvas

Working together over shared state.

Much of the **context problem** is really a **UI problem**. Good interfaces let end-users provide context and steer the agent effectively.

Takeaways:

- Don’t treat UI as an afterthought! It shapes how agents perform.
- Use the right modality (chat, canvas, generative UI) for the task.
- Empower users to inject context and guide the agent.

12. Situational Awareness

We are still **early** in the lifecycle of agentic applications.

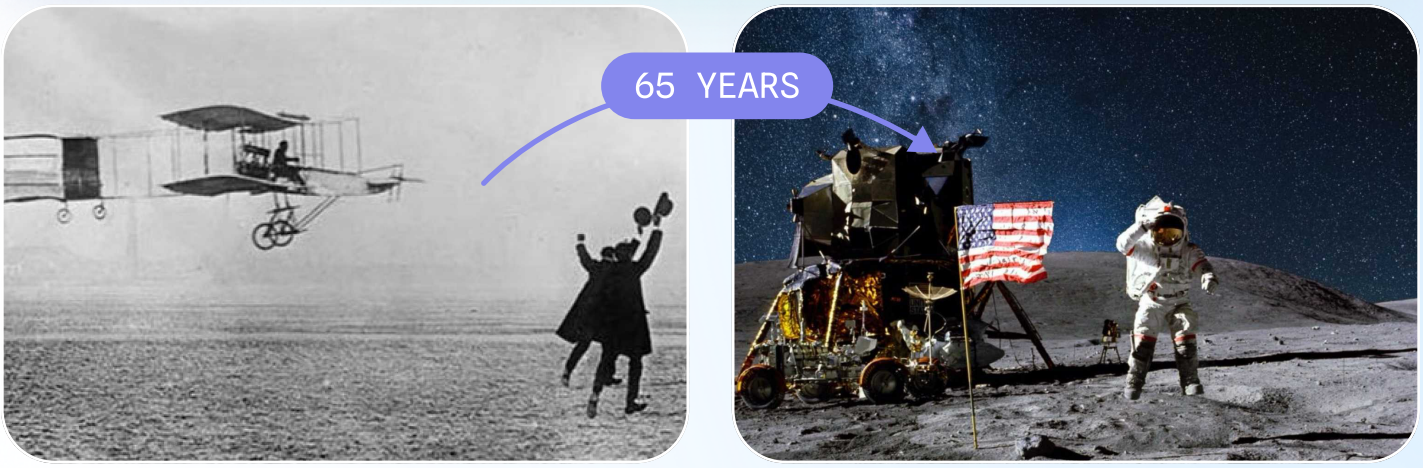
- Some of today's problems will disappear quickly, so don't over-invest in solving them.
- Expect major improvements at both the **model layer** and the **application layer.**
- The key is not to optimize for the present, but to **bet on the right direction.**

Be aware of forces shaping the future:

1. **Growing context size**
2. **Token costs**
3. **Long-running agents**
4. **Voice modalities**
5. **Reinforcement learning**

Takeaways:

- Don't waste cycles solving temporary limitations.
- Anticipate rapid progress in both infra and UX.
- Focus on long-term direction, not short-term fixes.



Once humanity figures out the fundamentals of some discipline, expect very rapid progress and investment over the ensuing decades.

Predictions

1. Most knowledge workers will spend the majority of their productivity time working **alongside AI agents.**
2. **Reinforcement learning** will scale and become central to agent improvement.
3. The **application layer** (UI + orchestration) will reign supreme.
4. **Voice** will grow as a critical modality.

So what is AG-UI and how does it help?

TECHNICALLY

An open, lightweight, event-based protocol for rich, real-time agent ↔ user interactivity.

PRACTICALLY

It lets you **seamlessly bring AI agents into user-facing applications.**

AG-UI Compatible Frameworks:

Takeaways:

- Expect rapid shifts in how users and agents collaborate.
- The real frontier is at the **application layer.**
- **AG-UI** provides the missing standard for agent-user interaction.

CopilotKit

Found this useful?

Repost It

to help one person in your network