ChatGPT

Designing AI Agents: Best Practices and Intent Handling

Understanding AI Agents and Intent

AI Agents Defined: An AI agent is an autonomous software entity that perceives its environment, reasons about it, and takes actions to achieve specific goals [1]. Modern agents, especially those based on large language models (LLMs), combine the language understanding and reasoning of LLMs with tools or APIs to act on the world [2]. In other words, an LLM-based agent isn't just a chatbot answering questions – it can **plan steps and execute tasks** (e.g. by calling external services) in pursuit of a user's intent. This bridges the gap between **reasoning** and **action**, allowing agents to carry out complex goals rather than just output text [2].

Intent in this Context: The *user's intent* is the goal or task the user wants the agent to accomplish. Designing effective agents means focusing on how the user expresses their intent and how the agent interprets and fulfills that intent. The interaction should feel like collaborating with a capable assistant, not just issuing a command to a passive program. The agent needs to understand **what the user really wants** (even if initially stated vaguely) and stay aligned with the user's objectives and values as it works [3].

Defining the Agent's Role and Scope

Identify the Right Use Case: A critical first step is deciding *what kind of agent is needed* and **where it fits in your process or workflow**. Not every problem requires an AI agent. As a rule of thumb, if a task is straightforward enough to be handled by a fixed **rule-based automation or a simple script, it likely doesn't need an AI agent** [4]. For example, if you can fully capture a process in a clear flowchart with no ambiguity or judgment calls, traditional automation is more efficient and predictable [5]. On the other hand, **good candidates for AI agents** are tasks or processes that:

- Involve **complex decision logic or multiple variables** that aren't easily codified into simple rules (e.g. handling various customer support scenarios with lots of exceptions) [6].
- Have **shifting context or many edge cases** where a static program would fail – for instance, workflows that would require a tangled web of if-then rules [6].
- Require understanding **unstructured data** like natural language text, images, or audio (for example, reading and summarizing documents, reviewing legal contracts, analyzing images) [7].
- Benefit from some level of **judgment or prioritization** that a human could do, but you want to automate (for instance, triaging support tickets or approving refunds with varying conditions).

At the same time, it's wise to **avoid fully automating truly high-stakes decisions** – those with serious legal, safety, or financial implications – unless you keep a human in the loop. If an error in judgment would be extremely costly or if regulations demand explainability, the agent's role should be limited to assisting a human decision-maker, not replacing them [8]. For example, one organization deliberately did *not* use an agent to make actual investment decisions (a high-risk task requiring human judgment); instead, they deployed agents for supporting tasks like conducting due diligence research and drafting content, leaving final critical decisions to people [9]. In short, **define clear boundaries** for the agent's domain.

Single Purpose vs. Generalist: As you identify the agent's role, try to give it a **clear, focused mission**. Each AI agent should have a well-defined purpose or set of responsibilities (often called the "single responsibility principle") [10] . For instance, you might design one agent specifically to manage meeting scheduling, and another agent to handle IT support queries, rather than one monolithic agent that does everything. Focused agents are easier to build, maintain, and align with user expectations [10] . They can also be more transparent about their capabilities. If you do need a very broad range of tasks handled, consider structuring this as **multiple specialized agents** that collaborate, rather than one unwieldy agent. (In complex systems, you might have a *planner* agent that breaks a high-level goal into sub-tasks, then delegate those to expert sub-agents like a "Retriever" agent for information gathering, an "Executor" agent for performing actions, etc. [11] – but all of that should still be conceptually clear to the user and designer.)

Fitting the Agent into the Process: To integrate an agent into real-world workflows, map out how it will **receive input and deliver output** in the context of user or business processes. Ask questions like: *How will the agent be triggered or invoked?* It could be via a chat interface, voice command, a button in an app, or an automated trigger (e.g. an agent that runs when a new email arrives) [12] . Also consider *what the agent will do when finished* – does it produce a report or recommendation for a human to review, does it automatically execute changes in a system, or hand off to another process? Design the agent so that its **hand-offs are clear**. For example, an agent might draft an email for you, but leave the final send decision to you; or it might autonomously place an order but log its actions for later review. Ensuring the agent's outputs are **transparent and verifiable** helps it fit smoothly into existing processes. Many successful deployments start small: identify a specific pain point or task in the workflow, build an agent to handle that, and measure results before expanding. A systematic approach can be useful:

1. **Survey & Select:** Catalog your current workflows and tasks (especially those with heavy manual effort or long delays). Highlight the pain points or bottlenecks [13] . Then evaluate each for complexity and suitability – eliminate tasks that are trivially simple (better done with standard automation) or too critical to entrust to AI [14] .
2. **Feasibility Check:** For the promising candidate tasks, assess what data, tools, or integration would be required. Make sure the agent can be given the information it needs (access to databases, APIs, documents, etc.), and check for any organizational or technical constraints [15] .
3. **Define Success & ROI:** Clarify what a successful outcome looks like (e.g. reducing response time by X, or achieving a certain accuracy), and weigh the benefits against the effort. This helps in setting the agent's goals and deciding how autonomous it should be [16] .
4. **Start Small:** Implement the agent in a limited, low-risk scenario first. Treat it as a pilot where you and the users can learn and adjust. Use clear metrics to evaluate its performance. This incremental approach ensures the agent truly fits into the process and provides value before scaling up [17] .

By carefully **choosing the agent's scope** and how it plugs into the workflow, you ensure the agent addresses a real need and operates within safe, well-understood limits. This upfront planning is key to a successful agent design.

Designing Agent Capabilities and Decision Boundaries

Once the agent's role is defined, the next step is to design *what it can and cannot do*. This involves deciding the **tasks it will handle autonomously, the tools or functions it will use**, and how it will make decisions or defer to humans. Some best practices and conceptual guidelines include:

- **Define Clear Capabilities:** List the specific actions the agent is allowed to take on the user's behalf. For an LLM-based agent, this often means defining a set of **tools or APIs** the agent can invoke. For example, a travel-booking agent might be equipped with a flight search API and a payment API; a coding agent might have tools to read from a code repository and write to it. By explicitly granting an agent certain tools, you both empower it and constrain it. Each tool should perform a *clear, deterministic operation* (like "query database" or "send email") without making high-level decisions [18] [19]. The agent's job is then to decide *which* tools to use and *when*, in order to fulfill the user's intent. This separation is a key design principle: let tools handle the low-level actions, and let the agent handle the **interpretation, strategy, and decision-making** based on tool outputs [19]. For instance, an agent might decide "I should look up the user's calendar now" (agent's reasoning) and then call a calendar API (tool execution). This keeps the agent's reasoning logic distinct from execution logic, making the system more modular and controllable [19].

- **Set Boundaries and Constraints:** It's crucial to establish what the agent *must not* do. This includes hard constraints (like "don't spend over $100 without approval" for a purchasing agent) and ethical or safety limits (like "never delete data unless explicitly confirmed by the user"). Providing these constraints as part of the agent's instructions or design is how we avoid the "genie problem" of an agent taking a directive too literally and causing harm [20] [21]. In other words, when you express the agent's goal, also express the guardrails: the values or rules it should uphold. For example, if the goal is to "increase website traffic", a well-designed agent should also be told (or constrained by design) *not* to pursue unethical or destructive methods. Otherwise, an unconstrained agent might technically satisfy the goal in an undesirable way (akin to a genie fulfilling a wish but with twisted results) [20]. A real design principle here is **value alignment** – ensure the agent's optimization criteria include the *spirit* of the user's intent and organizational values, not just the raw goal [21].

- **Decision Authority vs. Human Oversight:** Decide which decisions the agent can make autonomously and which require human confirmation or input. A robust conceptual approach is to give the agent autonomy for routine, low-risk decisions, but have it loop the human in for anything sensitive or uncertain. In practice, an agent might **pause and ask for user approval** at certain checkpoints if it's about to do something irreversible or high-impact [22]. For example, an email drafting agent might autonomously draft replies and even send routine ones, but for an unusual or high-importance email it could ask the user "Do you want me to send this response now?" before acting. This kind of *adaptive autonomy* keeps the user in control of critical decisions. It's also beneficial to design the agent to **explain its reasoning or plan** before executing major actions – for instance, "I plan to do X, then Y, because … – shall I proceed?" Making the agent's thought process explicit (a concept called *explicit reasoning transparency*) is a best practice for trust and debugging [10]. The user (or developer) should be able to understand *why* the agent is choosing a certain path. This transparency acts as a safety check and builds user confidence.

- **Memory and Context:** Include mechanisms for the agent to **remember context** across steps or interactions. Conceptually, *memory* means the agent retains relevant information it has observed or been told, rather than forgetting everything each turn. This can be as simple as the agent

keeping a history of the conversation, or as elaborate as integrating with a knowledge base. Memory is *crucial for context*, allowing the agent to recall prior details, user preferences, or intermediate results so it doesn't repeat questions or lose track of the goal [23]. For example, if earlier in the session the user said "By the way, my budget is $500", a well-designed agent will remember that constraint when making recommendations later. On a conceptual level, treating past interactions as an immutable log or "event memory" can help the agent maintain a coherent dialogue and plan [24]. This memory design ties into the agent's decision boundary: the agent should know when to rely on memory versus when to ask the user again (if information might be outdated or uncertain).

- **Testing and Guardrails:** Because an agent operates with some autonomy and can make chained decisions, it's important to have **guardrails and thorough testing** in place before deploying it widely. Conceptually, guardrails can include things like: limiting the number of actions the agent can take in a session (to prevent infinite loops or runaway behavior), sanity-checking the agent's outcomes against known constraints, and having fallback behaviors if something seems off-track. For example, an agent might be designed to stop and alert a human operator if it hits a certain threshold of uncertainty or a potential error state. Research and industry experience recommend testing agents in sandboxed or safe environments to see how they behave, especially for long or complex tasks [25]. This lets designers refine the agent's rules and knowledge before real-world use. Maintaining logs of the agent's decisions (an "immutable event log" of what the agent did and why) is also a best practice for observability and debugging [26] [27]. In sum, define **how far the agent's autonomy goes**, and put safety nets in place so it remains a reliable helper rather than a loose cannon.

By carefully designing the agent's capabilities and limits, you create a framework where the agent can be creative and useful **within bounds**. The agent knows *what tools it can use*, *what goals to aim for*, and *what rules not to break*. This upfront design of the agent's "decision space" makes the difference between a chaotic AI and a trustworthy assistant.

Expressing User Intent to the Agent

Designing an AI agent also involves designing **how the user will communicate their intent**. In LLM-based agents, the interaction is typically conversational – the user provides instructions or questions in natural language, and the agent responds in kind. This represents a shift from traditional software: instead of clicking buttons or filling forms in a GUI, **the user can simply tell the agent what they want in plain language** [28]. As designers, we need to ensure this natural language interface effectively captures the user's intent. Some considerations:

- **Natural Language Input:** Allow the user to express goals or requests in flexible, human terms. The user might say, *"Schedule a meeting with the marketing team for next week and prepare an agenda"*. The agent should be able to parse this and identify the intent (e.g., set up a meeting, coordinate calendars, draft an agenda). Using an LLM at the core helps because it can understand nuances and context in free-form language. However, **don't assume the first phrasing is complete** – people often state their intent at a high level, leaving out details. The system should be ready to handle that, which leads to the next point: the agent prompting *the user* for clarification.

- **Encouraging Clarity:** As part of the user experience, you might guide users to provide sufficient detail. This can be done passively (through UI hints or example commands shown to the user) or actively (with the agent asking follow-up questions). Users may not initially know what

information the agent needs. For example, if a user just says “Book me a flight,” the agent will need more details (to where? when? budget?). A well-designed agent will *prompt the user for those specifics* rather than either guessing or giving up. In essence, the design should treat the conversation like a collaborative planning session rather than a one-off command. Many mistakes and misinterpretations can be avoided simply by capturing the intent in more detail. In fact, ambiguous instructions are a known source of AI “doing the wrong thing” – if a request is too vague or underspecified, the agent might technically fulfill it in an unintended way [20]. To prevent that, the system should **invite the user to clarify their intent** in their own words. This might involve the agent asking questions like, “Could you tell me more about what you’re looking for?” or “What does success look like for you in this task?” before it proceeds. The goal is to *extract the true intent* behind the user’s initial request.

- **Intent Formats:** In some cases, especially in professional or complex settings, you might incorporate structured input alongside natural language. Conceptually, this could mean the agent supports both a free-form instruction and optional parameters or templates. For instance, a project-management agent might let advanced users fill in a brief template like: *Task: “Prepare monthly report”*, *Deadline: “Sept 30”*, *Priority: “High”*. However, even in these cases, the agent should handle natural language gracefully (e.g. it should equally handle “Hey, please prepare the September performance report by end of month”). The guiding principle is to **make it easy for the user to express what they want**, without forcing them to adapt to machine-speak. The onus is on the agent’s design to interpret human intent, not on the human to craft the perfect command. Techniques like robust natural language understanding or using the LLM’s prompt to restate the user’s request in a more structured form internally can help bridge this gap.

- **Providing Context to the Agent:** When expressing intent, it’s often helpful if the user (or the interface) provides relevant context up front. For example, “Draft an email to my manager about the project update” could be enhanced if the agent already knows who “my manager” is and what project is being referenced. In design terms, this means the agent may have access to user profiles, settings, or history that provide implicit context. The user’s intent expression can thus be shorter because the agent *remembers* or can look up supporting details. A best practice is to let users know what context the agent has (“Using your calendar and contacts on file”) so they can trust the agent to fill in blanks, and also to avoid surprises. The agent’s memory of prior interactions or stored user preferences plays a role here, ensuring that the intent is understood in light of past communications (for instance, understanding “my manager” refers to the same person discussed earlier).

In summary, expressing intent to an AI agent should feel natural and flexible. The design should minimize the burden on the user to phrase things “just right.” The user says what they want in their own words, and the agent is equipped (through its LLM and prompts) to interpret that. When details are missing, the agent doesn’t make blind assumptions – it engages the user to refine the intent. This brings us to the next crucial aspect: how the agent asks questions and collaborates to clarify intent.

Agent Clarification and Interactive Dialogue

A hallmark of a well-designed AI agent is its ability to engage in a **two-way conversation** to nail down what the user really wants. Instead of a one-shot request followed by a one-shot answer, the agent should behave more like a collaborative partner that **asks questions, confirms understanding, and**

iterates. This consultative interaction style is key to alignment and user satisfaction [3]. Here's how an agent can handle intent clarification conceptually:

- **Active Listening and Ambiguity Detection:** The agent should "listen" to the user's request and *detect if anything is ambiguous or missing*. Modern LLMs can be prompted to identify unclear aspects of a query. For example, if the user says "Help me improve my productivity," the agent can recognize that *"improve productivity"* is vague – productivity in what sense? work tasks? personal habits? what timeframe or tools? Rather than guessing, the agent employs **active listening**: it considers what was said and even what wasn't said [29] [30]. It might respond: *"Sure! To make sure I help effectively, could you clarify which areas you want to focus on? For example, managing your schedule, reducing distractions, or something else?"* [30]. This kind of question is open-ended and invites the user to elaborate. The agent essentially says "I'm here to help, I just need a bit more detail." This mirrors how a good human assistant would react rather than blindly acting on incomplete instructions.

- **Asking Open-Ended, Probing Questions:** When the agent needs more information, the form of its questions matters. **Open-ended questions** (those that can't be answered with a simple yes/no) are highly effective for gathering intent [31] [32]. They prompt the user to explain their needs or preferences in their own words. For instance, if a user tells a task manager agent "I need you to handle all my emails," a poor agent might either do nothing useful or take an overly literal (and possibly incorrect) action. A well-designed agent, however, will ask **clarifying questions**: *"Absolutely, I can help with email. What aspects of your email are you looking to automate or simplify?"* and follow-ups like *"Are there certain types of emails you want me to respond to, or specific contacts you prefer to handle personally?"* [33]. Questions like these encourage the user to spell out their priorities and constraints. It turns a broad intent (*"handle my emails"*) into a specific set of tasks the agent can actually perform (e.g. *"draft replies for meeting requests that aren't urgent, and flag important emails from my boss"*). A useful heuristic here is to start with broad questions and then get more specific based on the user's answers [34]. The agent might begin with "Tell me more about X" and then drill down into details like who, what, when, how often, etc., as needed [32]. This approach mimics how a skilled interviewer or consultant gathers requirements: zoom out, then zoom in.

- **Confirming Understanding (Paraphrase & Confirm):** After one or a few rounds of questions, the agent should play back its understanding of the user's intent to ensure nothing is lost in translation. This is the *clarify and confirm* step. The agent can **paraphrase** the user's requests and ask for confirmation: *"So, just to confirm, you'd like me to decline non-urgent meeting requests and highlight any emails from your direct reports – did I get that right?"* [35]. By rephrasing the goals in its own words, the agent not only double-checks accuracy but also gives the user a chance to hear how the agent interpreted their intent. The user might catch a misunderstanding at this stage ("Actually, no, I meant something slightly different..."), which is far better than catching it after the agent has acted incorrectly. This technique of summarizing and confirming is borrowed from effective human communication, and it's invaluable in agent design for aligning on intent early [35]. It builds trust, because the user sees the agent is not rushing to action without their approval.

- **Empathy and Tone:** The way an agent asks for intent or clarification can influence user comfort. A purely robotic or terse tone may make the user reluctant to engage in a back-and-forth. A more **empathetic, conversational tone** helps the user feel the agent is a partner rather than a pedantic machine [36] [37]. Simple phrases like "I want to make sure I get this right for you..." or "No problem, I just need a bit more detail to help with that" can reassure the user that the agent is working *with* them, not just giving them chores to do [37]. This doesn't mean the agent needs

to simulate human emotions, but it should follow basic conversational etiquette and friendliness. Being too formal or abrupt could hinder the collaborative atmosphere. In conceptual design, defining a personality or tone guide for the agent can be useful – for example, *helpful, patient, and professional*. This ensures the agent's prompts for intent clarification are well-received and not misconstrued as errors or annoyances.

- **Iterative Refinement:** Importantly, **intent clarification is not necessarily one-and-done**. The agent may need to engage in a short dialogue, asking a couple of questions one after another, to fully scope the task. Both the user and agent should expect an *iterative process*, much like having a brief conversation with a human assistant. For complex requests, the agent might propose a plan and then allow the user to refine it. Embracing an **iterative workflow** leads to better outcomes [38]. For example, if the goal is complex ("Plan my entire marketing campaign"), the agent might break it down and say: *"Okay, I can draft a plan that includes social media, email, and event marketing. Let's start with social media – would you like to focus on any particular platform or audience?"*. The user and agent might go back and forth on one subtask, then move to the next. This is far more robust than expecting the user's initial one-sentence request to yield a perfect, final result in one step [38] [39]. The **conversation-like interaction** means the agent can course-correct as new information comes up, and the user can iteratively steer the agent.

- **Convergence and Execution:** After sufficient clarification, the agent should summarize the agreed-upon intent or plan and proceed to execute the tasks within its scope. Because of the dialogue, by this point the agent's actions are well-defined and much more likely to satisfy the user's true intent [40]. The user, having been involved in setting the parameters, has a mental model of what the agent will do. This dramatically increases the chance that the outcome matches what the user actually wanted [41]. If during execution the agent encounters something unexpected or a decision not covered in the clarification phase, a good design is for the agent to loop the user back in: essentially, *request intent for the new sub-problem*. For instance, if while handling your emails the agent finds an invite that it's unsure to accept or decline, it might ask: "I've found an invitation to X. Do you want me to accept it, decline it, or leave it for you to decide?" This keeps the interaction going as needed, ensuring the agent never goes too far off-track on its own.

In essence, the agent's ability to **request intent** (ask questions) and the user's ability to refine their instructions are two sides of a cooperative conversation. This design paradigm – treating AI-agent interaction more like a **collaborative dialogue** rather than a one-way command execution – is fundamental to building safe and effective agents. It prevents the "wish-granting genie" scenario by turning the agent into a **guide and partner** that *first seeks to understand, then to be understood*. By the time the agent takes action, it's operating on a well-specified intent, greatly reducing surprises [41].

Conclusion: Conceptual Best Practices Recap

Designing AI agents at a conceptual level involves balancing **user-friendly interaction** with **well-structured agent behavior**. To recap the key best practices:

- **Clear Purpose & Scope:** Define exactly what the agent is for and **choose the right tasks** for it. It should tackle complex or data-intensive problems that benefit from AI reasoning, not trivial or overly risky tasks better left to humans or simple software [8] [6]. Keep each agent's role focused so it's understandable and manageable [10].

- **Capabilities with Boundaries:** Empower the agent with the tools and knowledge it needs, but also **impose boundaries**. Specify what actions it can take and embed constraints so it pursues goals in an acceptable way [21]. Design the agent to use tools for discrete actions while it handles the higher-level decision-making [19]. Always decide which decisions it can make alone and where a human should be involved [8] [22].

- **Intent Communication is Two-Way:** Make it easy for users to **express their intent** in natural language [28], and design the agent to actively **clarify that intent** through dialogue [30] [35]. The agent should ask open-ended questions, confirm it understands the request, and iterate with the user until the task is clearly defined. This conversational approach keeps the agent aligned with what the user actually wants at each step [3].

- **Interactive and Transparent Workflow:** Treat the agent's operation as an ongoing interaction. The agent can share its **thought process or plan** with the user (in a concise way) to be transparent [10]. It should also have the ability to pause, take feedback, or adjust its course based on user input or new information during execution [22]. This makes the user feel in control and builds trust.

- **Testing and Evolution:** Finally, think of agent design as iterative. Start with a solid conceptual design, but be prepared to refine it by observing real interactions. Monitor how well the agent is interpreting intents and where misunderstandings occur. Use those insights to improve the agent's prompts, its clarification questions, or its constraints. Over time, the agent can be expanded with more capabilities or even cooperate with other agents, but always on top of a well-defined core that keeps it reliable. A simple, transparent design often outperforms an overly complex one in real-world settings [42]. Prioritize a design where you (and the users) can **understand and trust the agent's decisions** at a high level, since that trust is what ultimately allows an AI agent to become a valuable assistant in any process.

By following these conceptual best practices, one can design AI agents that are **intent-aligned, responsive, and safe**. The agent effectively becomes a digital team-mate – you give it a goal, it asks the right questions, and then takes initiative to carry out tasks, all while keeping you in the loop. This collaboration between human intent and AI autonomy is at the heart of modern AI agent design. With clarity of purpose and thoughtful interaction design, an AI agent can truly fit into your process and amplify what you can achieve.

Sources: The principles above are synthesized from emerging best practices in AI agent design, including research and expert guidance on agentic systems [2] [19] [3] [30] [8], which emphasize clear goals, interactive clarification, and well-bounded autonomy for AI agents.

---

[1] [24] [26] [27] Design LLM-Based Agents: Key Principles — Part 2 | by Craig Li, Ph.D | Binome | Medium
https://medium.com/binome/design-llm-based-agents-key-principles-part-2-8f4011e54637

[2] [10] [18] [19] The Definitive Guide to Designing Effective Agentic AI Systems | by Manav Gupta | Medium
https://medium.com/@manavg/the-definitive-guide-to-designing-effective-agentic-ai-systems-4c7c559c3ab3

[3] [20] [21] [29] [30] [31] [32] [33] [34] [35] [36] [37] [38] [39] [40] [41] Avoiding the "3 Wishes" Problem in Agentic AI Design
https://blog.agent.ai/avoiding-the-3-wishes-problem-in-agentic-ai-design

4 5 6 7 8 9 12 13 14 15 16 17 The Ultimate Guide to Designing And Building AI Agents - Sid Bharath
https://www.siddharthbharath.com/ultimate-guide-ai-agents/

11 23 28 Building effective AI agents: A brief guide and best practices | by Anastasios (Tasos) Stamoulakatos | Satalia AI Research Lab | Aug, 2025 | Medium
https://medium.com/wpp-ai-research-labs/building-effective-ai-agents-a-guide-to-the-future-of-llms-and-our-proposed-best-practices-06f6ca7c250f

22 25 Building Effective AI Agents \ Anthropic
https://www.anthropic.com/engineering/building-effective-agents

42 Building Effective AI Agents - Anthropic
https://www.anthropic.com/research/building-effective-agents