1. Select an agent framework (autogen (multiple agents), agentsdk (one agent high abstraction low control) or one agent low abstraction high control)

2. Construct goal tree.

3. Assign resource bounds.

For each goal:

Plan

Is the problem well defined? No? ask user or do research on trusted sources to better understand. Can't find enough info to reliably proceed? Stop and wait for user input. Otherwise if in auto mode create as much value as possible and skip to the next part of the task that is highest yield and which we can do next. If there is no remaining value or stuck, stop ask for feedback and wait before proceeding. Otherwise, if we'll defined and trusted source + good understanding (many worked examples to test understanding on) proceed. Not enough examples - as user. Analyse task atomicity - too hard? Decompose. Repeat planning loop. Set constraints.

4. Select tools.

5. Select workflow.

6. Get context.

7. Get RAG data/research.

Do a final check - do we have a clear idea of what steps need to be undertaken to complete the problem AND do we think the problem can be satisfactorily resolved AND do we have a way of evaluating the end solution. If so then yes, please proceed to solve.

8. Solve.

9. Review.

Forward dependencies if task completed. For each step stop and above loop to safely proceed! CSF framework to guide.

Step up to larger goal in goal tree on success /down to decompose to smaller problems if the problem is too hard/cycle if the problem is solvable but needs more effort - do a root cause analysis for failure and try to correct this in the next cycle based on eval + review step progress in the context of the rest of the problem being solved.

Repeat until progress can't be made, resources exhausted or safety error.

Sometimes you just have to learn by doing - if the system can't tick all the boxes ideally just try and see how it goes to get more info if the user cannot provide more info (unless it truly is a fool’s errand? Then warn user, chances of successare very low... and proceed)

Emit tasks during workflow for human verification.

Simple 5 part UI - chat agent (regular chat interface between enhanced with memory, reasoning, learning and agentic abilities), worker agent (a play pause button with continuous trace which is a thought and associated action tuple which is given as a ‘continuous stream of consciousness’ with a relatively low level of detail, guardrails and constraints on usage), telemetry (tokens/min, total cost including breakdown by task, system memory, time of execution etc for agents + what system has learned during the solving process - explicit memories and tool tracking), assets (all the files which have been created by the agent).

We are solving the:

1. Deployment problem: people shouldn't need a master's in AI to be able to get value from AI to do their job.

2. Value problem: it shouldn't be so hard to get demonstrable value from AI I.e. SME and even large enterprise shouldn't have to build their own custom solutions, end users should have one tool like ChatGPT but tailored to them and fully agentic/capable like a real worker and not a chatbot.

3. Using, creating and testing/debugging tools in LLMs is still a grey area even for fairly grey area for many devs let alone laypeople. Let's make this simple - give your agent skills and let your agent learn over time.

4. Let's make the agent experience more frictionless. I want to set and forget so I can spend more time doing the work I want and trust the agent is going to stay on task, aligned and not do anything stupid.

5. People don't need another generic tool, they need a tool which is tailored to their needs that they can trust to do the job safely so they don't need to double check and baby the AI taking longer to do the same job than doing it themselves.

6. We need AI workers rather than ping pong chat to get long context large scale task value delivery which is much closer to what human workers do.

7. The intuition? Use AI to make AI easier to use - an operating system for AI not an AI operating system. Most llms are a tool without a problem - the solution is to help the user find the problem first and then solve it intelligently so they get maximum bang for their buck without needing to learn how to use AI. Just a lot more user friendly.

Todo today- build docs, build pipeline spec + algorithm and stubs with psuedocode, design v0 ui.

Training by example the human does a task while explaining why/how in order to train the AI.

This goes both ways and the AI can produce worked examples so you can verify and give feedback and reattempt.

Or AI can teach human or another AI.

ibm-granite/granite-docling-258M

Best in class for open source equation diagram etc to text.

Step-by-step Specification

1) Select an agent framework

Inputs: task complexity, need for collaboration, safety gating, observability, deployment constraints.

Output: framework choice + rationale + instrumentation hooks.

2) Construct goal tree

Inputs: user objectives; inferred subgoals.

Output: prioritized tree with acceptance criteria and dependencies.

3) Assign resource bounds

Inputs: org policy, cost/time limits, sensitivity levels.

Output: enforceable limits + escalation rules (ask/stop).

For each goal

Plan:

Check well-definedness; if ambiguous, ask or research.

If info insufficient, stop and wait.

If Auto mode: proceed to next highest-yield actionable subtask that is safe.

Require trusted sources + worked examples to validate understanding; if lacking, ask user for examples.

Assess atomicity; if too hard, decompose and loop.

Set constraints: acceptance tests, KPIs, safety checks, rollback.

Select tools: map subtasks to tools; ensure sandboxing, access, and telemetry enabled.

Select workflow: single vs multi-agent, parallelism, review gates, human-in-the-loop points.

Get context: workspace, memory, prior artifacts, environment; detect conflicts.

Get RAG/research: retrieve/reason over trusted sources; maintain citations.

Final check (go/no-go): steps known, solvable under bounds, evaluable end state.

Solve: execute with guardrails; emit human verification tasks at checkpoints.

Review: run tests/linters/evals; validate against acceptance criteria; gather evidence.

Forward dependencies: unlock parent/siblings; update goal tree.

Adaptive control: step up (success), step down (decompose), or cycle (RCA and retry).

Repeat until: no progress possible, resources exhausted, or safety error.

Learning-by-doing fallback

If info is partial but action is safe: attempt smallest reversible step to gain signal.

Warn user if success likelihood is very low; proceed only if low-risk and bounded.

Human verification tasks

Emit approval tasks at: ambiguous requirements, external write actions, policy gates, large resource escalations, low-confidence decisions.

Safety and Governance

Guardrails

Policy enforcement, tool access control, data handling, external action whitelists, rate/size limits, sandboxing, content filters, rollback plans.

Evaluation

Predefined acceptance tests, heuristic checks, regression tests, human review gates, cost/benefit tracking.

Stop conditions

Ambiguity not resolvable, unsafe action, exhausted bounds, failing evaluations without feasible correction.

Telemetry and Learning

Telemetry

Tokens/min, total tokens, latency, cost by task/subtask, memory usage, tool usage, error rates, confidence scores.

Learning

Explicit memories (facts/preferences), episodic traces, tool success/failure stats, example banks, RCA outcomes feeding future planning.

Five-part UI

Chat agent

Rich chat; remembers preferences; shows citations; supports “explain/rationale on demand.”

Worker agent

Play/Pause; continuous trace stream: thought + action tuples; constrained verbosity; guardrails visible; step rollback.

Telemetry

Live metrics (tokens/min, cost breakdown, time); system memory summaries; learning log.

Assets

Artifact explorer of created/edited files; diffs; version history; approvals.

Controls

Mode toggle (Auto/Assisted), resource bounds panel, safety policies, review gates, approvals queue.

Clear Examples

Example A: Deployment problem (SME support agent)

Framework: agentsdk (single-agent, high abstraction).

Goal tree: G0 “Deploy support agent”

G1 “Ingest FAQ + KB”

G2 “Setup email + chat integration”

G3 “Define guardrails + escalation”

G4 “Pilot with 10 tickets; success ≥85% CSAT”

Bounds: $30/day, ≤1h setup, no PII exfiltration, external write actions require approval.

Tools: Email API (read-only first), Helpdesk API sandbox, RAG on KB.

Final check: steps known; solvable; CSAT eval defined.

Human tasks: approve integration scopes; approve pilot start.

Outcome: deploy pilot, review metrics, forward-depend to full rollout.

Example B: Value problem (weekly exec report)

Framework: autogen (analyst + reviewer).

Goal tree: collect data → synthesize insights → draft → review → deliver.

If “well-defined?” = No: ask for KPIs + audience; research trusted benchmarks.

Worked examples: prior reports; request from user if missing.

Bounds: 20 minutes, $2, no external writes.

Human tasks: approve KPI list; approve final draft.

Outcome: consistent, tailored, evaluable deliverable.

Example C: Tool creation/testing (web scraper)

Framework: low-abstraction single agent for fine control.

Plan: generate minimal prototype; test on 3 sample pages; log failures.

Learning-by-doing: safe dry-run; expand coverage based on results.

Bounds: 30 minutes, $1; respect robots.txt; no login.

Human tasks: approve target domains list; approve storage location.

Outcome: working tool + tests + telemetry; forward-depend to enrichment.

Example D: Frictionless long-horizon task (Auto mode)

Guardrails: approvals for external modifications; periodic summaries every N steps; auto-halt on low confidence or cost spike.

Emitted checkpoints: schema changes, third-party calls, policy-sensitive content.

Acceptance Criteria (per goal)

Clear definition and constraints set.

Trusted sources and worked examples documented.

Evaluations/tests defined and passed.

Artifacts stored in Assets with traceability.

Telemetry and cost within bounds.

Forward dependencies updated; no unresolved safety alerts.

User set checkpoints - if you get to this stage and can’t do this then stop - or ask. If you find that you are using too many tokens and get stuck then do stop and get user help etc.

—----------------------------------

flowchart TD

A["Start"] --> B["Select agent framework: autogen / agentsdk / low-abstraction"]

B --> C["Construct goal tree"]

C --> D["Assign resource bounds **(**time**,** cost**,** tokens**,** policies**)**"]

D --> E{"Goals remaining?"}

E -- No --> Z["Stop: No progress or all done"]

E -- Yes --> F["Select next goal by highest yield/feasibility"]

subgraph GoalLoop["Per-Goal Loop"]

F --> G["Plan"]

G --> H{"Problem well-defined?"}

H -- No --> I["Ask user OR Research trusted sources"]

I --> J{"Sufficient info?"}

J -- No --> K["Stop & wait for user input"]

J -- Yes --> L{"Auto mode?"}

L -- Yes --> M["Proceed to next highest-yield actionable subtask"]

L -- No --> N["Request minimal clarifications"]

H -- Yes --> O{"Trusted sources + many worked examples?"}

O -- No --> P["Gather examples / ask user for examples"]

O -- Yes --> Q["Analyze atomicity; too hard?"]

Q -- Yes --> R["Decompose; repeat planning"]

Q -- No --> S["Set constraints **(**quality**,** safety**,** acceptance**)**"]

S --> T["Select tools"]

T --> U["Select workflow **(**single vs multi-agent; steps/order**)**"]

U --> V["Get context **(**workspace**,** state**,** prior artifacts**)**"]

V --> W["Retrieve RAG/research data"]

W --> X{"Final check: clear steps? solvable? evaluable?"}

X -- No --> Y["Emit human tasks or fetch more info; possibly decompose"]

X -- Yes --> AA["Solve **(**execute plan**)**"]

AA --> AB["Review **(**eval criteria**,** tests**,** validation**)**"]

AB --> AC{"Passed?"}

AC -- Yes --> AD["Forward dependencies; mark goal done"]

AC -- No --> AE["Root cause analysis; adjust plan/constraints"]

AE --> AF{"Too hard?"}

AF -- Yes --> R

AF -- No --> AA

AD --> AG{"Step up to parent goal?"}

AG -- Yes --> F

AG -- No --> E

end

M --> S

N --> S

P --> S

Y --> F

K --> Z

R --> G