

THE RESEARCH LEDGER

AI Agents for Researchers

A Practical Guide to Literature Review,
Coding, Experiments, Writing,
and Reproducible Science

Ngo Trung Kien, PhD

First Edition

Contents

Visual Overview	xiii
Preface	xv
How to Use This Book	xix
Acknowledgments	xxi
About the Author	xxiii
List of Figures	xxxii
List of Tables	xxxvii
List of Listings	xlvi
 Part I Foundations of Agentic Research	 1
1 The Paradigm Shift: From Passive Tools to Autonomous Scientific Agents	3
1.1 Three Regimes of Machine Assistance in Science	4
1.2 What Makes an Agent	7
1.3 Grading Autonomy: The L0–L5 Ladder	10
1.4 Memory and Context	13
1.5 The RESEARCHOS Reference Architecture	15
1.6 The Delegation Contract and an Economics of Verification	21
1.7 What Agents Are Currently Bad At	25
Chapter Summary	27
Exercises and Discussion Questions	29
Further Reading	30

Part II Literature Review and Knowledge Synthesis 33

2 Agentic Literature Search and Citation Graph Navigation 35

2.1 The Hardest Easy Problem	36
2.2 Query Formulation as an Agent Task	38
2.3 Federated Retrieval	40
2.4 Deduplication and Canonicalisation	44
2.5 Citation Graph Navigation	47
2.6 Screening with Calibrated Thresholds	50
2.7 Measuring Search Quality	53
Chapter Summary	56
Exercises and Discussion Questions	57
Further Reading	58

3 Synthesising Evidence, Building Taxonomies, and Finding Gaps 61

3.1 From a Pile of Documents to an Evidence Table	62
3.2 Structured Extraction That Is Allowed to Refuse	64
3.3 PRISMA as an Executable Workflow	67
3.4 Inductive Taxonomy Construction and Its Validation	71
3.5 Gap Identification: Gaps, Nulls, and Contradictions	73
3.6 Quantitative Synthesis and When Not to Pool	75
3.7 Reporting the Review So It Can Be Re-Run	79
Chapter Summary	81
Exercises and Discussion Questions	82
Further Reading	83

Part III Coding and Research Software Engineering 85

4 Code Generation, Refactoring, and Research Software Engineering 87

4.1 Research Code Is a Different Artifact	88
4.2 Specification Before Generation	90
4.3 The Generate–Test–Repair Loop	94
4.4 Tests That Catch Scientific Bugs	98
4.5 Refactoring Inherited Analysis Scripts	102
4.6 Interfaces, Dependencies, and Environment Discipline	105
4.7 Reviewing Machine-Written Code	106

Chapter Summary	109
Exercises and Discussion Questions	110
Further Reading	111
5 Debugging, Profiling, and Automated Repair in Computational Pipelines	113
5.1 The Anatomy of a Scientific Bug	114
5.2 Reproduce Before You Repair	116
5.3 Reproducer Minimisation	120
5.4 Fault Localisation	122
5.5 Profiling and Performance	125
5.6 The Execution Sandbox	127
5.7 Regression Discipline and the Bug Ledger	129
Chapter Summary	133
Exercises and Discussion Questions	134
Further Reading	135
Part IV Experiments and Data Analysis	137
6 Hypothesis Generation, Experimental Design, and Parameter Sweeps	139
6.1 Where Hypotheses Come From	140
6.2 The Hypothesis Register	142
6.3 Design	144
6.4 Power Before Compute	148
6.5 Search Strategies	150
6.6 Orchestrating the Sweep	153
6.7 Interim Looks and Stopping	156
Chapter Summary	158
Exercises and Discussion Questions	159
Further Reading	160
7 Statistical Rigour, Data Validation, and Automated Exploratory Analysis	163
7.1 Automation Multiplies the Discipline You Already Had . .	164
7.2 Data Validation as a Gate	166
7.3 The Explore/Confirm Split	171

7.4	Choosing Tests and Reporting Effects	173
7.5	Multiplicity	175
7.6	Uncertainty When the Sampling Model Is Unclear	179
7.7	Making the Analysis Agent Auditable	182
	Chapter Summary	184
	Exercises and Discussion Questions	186
	Further Reading	187

Part V Scientific Writing and Peer Review 189

8 Multi-Agent Writing Systems: IMRaD Drafting and Citation Verification 191

8.1	What Writing Is For	192
8.2	Argument Skeletons Before Prose	194
8.3	Section-Specific Strategies	197
8.4	The Four-Gate Citation Verification Pipeline	200
8.5	Claim–Evidence Binding and the Traceability Matrix	206
8.6	The Writer–Critic Loop	208
8.7	Authorship, Disclosure, and the Non-Delegable Core	210
	Chapter Summary	212
	Exercises and Discussion Questions	213
	Further Reading	214

9 Automated Peer Review, Red-Teaming, and Quality Assurance 217

9.1	Pre-Submission Review as Engineering	218
9.2	Reviewer Agents with Distinct Lenses	220
9.3	Red-Teaming the Central Claim	223
9.4	Reproducibility Review	226
9.5	Aggregation and Prioritisation	228
9.6	Evaluating the Reviewers	231
9.7	Limits and Ethics in Real Editorial Processes	236
	Chapter Summary	238
	Exercises and Discussion Questions	240
	Further Reading	241

Part VI	Reproducibility, Safety, and Synthesis	243
10	Reproducibility, Containerisation, and Artifact Provenance	245
10.1	What Reproducibility Actually Requires	246
10.2	Environments: Lockfiles, Digests, and the Two-Year Test .	249
10.3	Data: Snapshots, Content Addresses, and FAIR in Practice	253
10.4	Determinism and Its Limits	255
10.5	Provenance Graphs and the Run Ledger	257
10.6	Packaging the Artifact	260
10.7	Reproducibility as Continuous Integration	263
	Chapter Summary	265
	Exercises and Discussion Questions	266
	Further Reading	268
11	Safety, Isolation, Threat Modelling, and Human-in-the-Loop Governance	269
11.1	The Threat Model	270
11.2	Indirect Prompt Injection Through the Literature	274
11.3	The Software Supply Chain	277
11.4	Containment: Least Privilege in Practice	278
11.5	Budgets and Circuit Breakers	281
11.6	Governance: The Four Gates as an Institutional Interface .	284
11.7	Institutional Policy	288
	Chapter Summary	290
	Exercises and Discussion Questions	291
	Further Reading	293
12	Capstone: An End-to-End Autonomous Research Assistant	295
12.1	The Study and Its Constraints	296
12.2	Stage-by-Stage Walkthrough	298
12.3	What Went Wrong	302
12.4	Where the Savings Actually Came From	305
12.5	Evaluating the Pipeline	307
12.6	The Agentic Research Maturity Model	309
12.7	What Is Coming and What Should Not Change	312
	Chapter Summary	314
	Exercises and Discussion Questions	315
	Further Reading	316

Part VII	Appendices	319
A	The Agent Protocol Library	321
A.1	The Task Contract, in Full	322
A.2	Role Protocols	325
A.3	Output Schemas	333
A.4	Verification Procedures	337
A.5	Escalation and Refusal	341
A.6	Anti-Patterns	344
B	Checklists, Templates, and Reporting Standards	347
B.1	Literature Search Protocol	348
B.2	Screening and Extraction Audit	349
B.3	Citation Verification Standard Operating Procedure	353
B.4	Pre-Registration Template for Computational and Machine-Learning Studies	355
B.5	Reproducibility Checklist	357
B.6	Artifact Manifest Specification	359
B.7	Safety and Containment Review	362
B.8	Disclosure Statement Templates	367
B.9	Institutional Policy Skeleton	368
C	Building RESEARCHOS: A Reference Implementation	373
C.1	Prerequisites and Project Layout	374
C.2	Environment: One Image, Two Places	376
C.3	The Kernel	379
C.4	Tool Adapters	384
C.5	The Task Graph Runner	386
C.6	Configuration and Policy as Data	389
C.7	Running the Capstone	391
C.8	Troubleshooting	393
	Glossary	397
	Bibliography	407
	Index	439

1

The Paradigm Shift: From Passive Tools to Autonomous Scientific Agents

Consider an ordinary week. A second-year doctoral student in machine learning — call her Mai — keeps an honest time log for seven days and totals 46 hours of work. Eleven of those hours are spent reading: skimming 63 abstracts returned by three overlapping database queries, downloading 19 PDFs, reading 6 of them properly, and discovering on Thursday that two of the six were superseded by a 2024 paper she had not found because it used “cosine annealing” where her query said “learning-rate decay”. Nine hours go to code: not to the interesting part, but to reconciling a decade-old repository’s pinned CUDA version with the cluster’s driver, rewriting a data loader whose original author has graduated, and adding the logging she should have added in week one. Six hours are spent waiting — queue time, checkpoint copying, a sweep that dies at 03:40 because a single dataset shard was truncated. Four hours go to formatting: bibliography entries with mismatched venue names, a table that will not fit the column width, a figure regenerated four times because the font sizes were illegible in print. Three hours go to meetings. And roughly thirteen hours — the residue — are spent on the thing she is actually paid to do: deciding what question is worth asking, judging whether a result is real, and choosing what to do next.

Now sort that week into two bins. In the first bin: work that is difficult but mechanical, where the criteria for success can be written down in advance and checked afterwards. Query expansion. Deduplicating a record set. Porting a data loader. Regenerating a figure at 300 dpi with 9-point labels. Re-running a sweep with a different seed. Reformatting 240 references to a journal’s style. Checking that every numeric claim in a draft appears in the results file. In the second bin: work where the criteria for success are themselves the object of inquiry. Whether the question is interesting.

Whether an effect of 0.4 percentage points across five seeds means anything. Whether a reviewer's objection is fatal or merely irritating. Whether the study, if it succeeds, deserves to exist. The first bin accounts for about 33 of Mai's 46 hours. The second bin is the research.

The thesis of this book is narrow and, I think, defensible: a large fraction of the first bin can now be delegated to software that plans, acts, observes the consequences of its actions, and revises — and the value of doing so depends almost entirely on whether the delegating researcher can check the result cheaply. That last clause is not a caveat. It is the engineering problem. A system that saves you nine hours of coding and costs you eleven hours of forensic verification has made your week worse, and it will have done so while producing artifacts that look more polished than anything you would have written yourself. Polish is not evidence. The discipline this book teaches is the discipline of building delegation that is cheap to audit.

This chapter lays foundations that every later chapter assumes. Section 1.1 places today's systems in a three-regime history of machine assistance in science, so that the current moment becomes legible as a shift in kind rather than merely in scale. Section 1.2 defines an agent precisely enough to argue about, distinguishing it from a chat interface by the presence of actions with consequences. Section 1.3 introduces a six-level autonomy ladder and locates the accountability boundary on it. Section 1.4 treats memory and context as a first-class architectural concern rather than a prompt-length inconvenience. Section 1.5 presents RESEARCHOS, the five-layer reference architecture that Chapters 2–11 each extend, and introduces RESAGENT, its reference deployment, together with the methodological meta-study that serves as this book's running scientific problem. Section 1.6 develops the delegation contract and its economics, deriving the condition under which delegation pays and the ceiling that verification cost imposes on any possible speed-up. Section 1.7 closes with a taxonomy of what these systems currently do badly, because a practitioner's guide that omits the failure modes is marketing.

1.1 Three Regimes of Machine Assistance in Science

It helps to see the present as the third of three regimes, each defined less by its algorithms than by the kind of relationship it establishes between the scientist and the machine. Figure 1.1 sketches the trajectory.

The *symbolic era* treated scientific reasoning as inference over explicitly represented knowledge. Its characteristic artifacts were *expert systems*

Machine Assistance in Science: 1965 to 2025

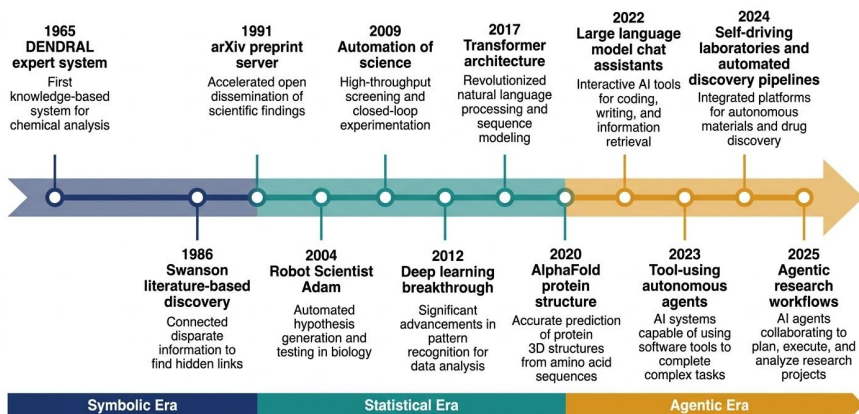


Figure 1.1: From symbolic inference to statistical prediction to agentic action. The first regime encoded human expertise as explicit rules and searched over them; its scientific triumphs were hypothesis-generating, as in literature-based discovery. The second regime replaced rules with learned functions and made prediction itself a research instrument. The third regime, the subject of this book, wraps such models in loops that select and execute actions — searching, writing code, launching jobs, editing manuscripts — and observe the results. The dates mark representative published anchors rather than sharp boundaries; the regimes overlap heavily and all three remain in active use today.

that encoded a specialist’s decision procedure as rules, and its most elegant scientific result was arguably *literature-based discovery*: Swanson’s demonstration that a novel, testable clinical hypothesis — a link between dietary fish oil and Raynaud’s syndrome — could be extracted from two disjoint bodies of published literature that no single reader had connected (Swanson, 1986). What matters about that work is its epistemic posture. The machine decided nothing; it surfaced a juxtaposition, and a human judged it. The regime’s later ambitions were larger. King et al. (2009) described closed-loop systems that formulate hypotheses about gene function, design the discriminating experiment, execute it with laboratory robotics, and revise — a genuinely autonomous scientific cycle achieved a decade and a half before “AI agent” entered the common vocabulary. It is worth remembering that the first working autonomous scientist was built with symbolic machinery and a pipetting robot, not a language model.

The *statistical era* shifted the locus of value from encoded rules to learned

functions, and in doing so made prediction a research instrument in its own right. A predictor accurate enough stops being a convenience and becomes an observation channel: protein structure prediction at near-experimental accuracy changed what questions structural biologists could ask (Jumper et al., 2021), and instrument-like predictors followed in drug discovery, materials screening, and plasma control, each collapsing a slow experimental loop into a fast computational one. Note the posture again. The model produced a number and a human decided what to do with it; the machine's authority extended no further than the accuracy of its output on a held-out set. Meanwhile the same era built the substrate for what came next. The transformer architecture (Vaswani et al., 2017) and the observation that sufficiently large language models perform unseen tasks from instructions and a handful of examples (Brown et al., 2020) together made it possible to write a general-purpose text-to-action interface without training a bespoke model for each action. Two further steps made that interface usable rather than merely possible: tuning on human preference data taught models to follow an instruction instead of continuing a passage of text (Ouyang et al., 2022), and the appearance of competitive open-weight foundation models put the same class of policy inside institutional infrastructure, where sensitive data can stay (Touvron et al., 2023). Capability growth across this period was tracked with broad multi-task suites — massed academic and professional examinations (Hendrycks et al., 2020) and the several hundred collaboratively contributed tasks of BIG-bench (Srivastava et al., 2022) — whose limitation, once agents arrived, was that they scored answers rather than actions. Wang et al. (2023b) survey the era's scientific reach with appropriate care. What the survey makes clear in aggregate is that the frontier had already moved: the interesting question was no longer how to predict one quantity well but how to amplify the discovery process itself.

The *agentic era* is defined by a change in what the system is permitted to do. In the first two regimes the machine emitted a symbol — a rule firing, a predicted probability, a paragraph — and a human decided what to do with it. In the third, the system selects and executes actions, observes what happens, and conditions its next action on that observation. The difference is consequential in the literal sense: actions have consequences, including irreversible ones. The capability that made the shift practical was reported in engineering documents rather than in methods papers: the GPT-4 report is organised around performance on professional and academic examinations

rather than around a training objective (OpenAI et al., 2023), and the Gemini report treats multimodal input as native rather than as an attachment (Team et al., 2023). Outside the sciences, two 2023 demonstrations set patterns that scientific systems then borrowed — a population of simulated characters driven by a memory stream with periodic reflection (Park et al., 2023), and an open-ended embodied agent that accumulated reusable skills over long horizons in a game environment (Wang et al., 2023a). Chemistry provides the clearest published demonstrations within science itself. Bran et al. (2023) wrapped a language model in a suite of chemistry tools so that synthesis planning became a sequence of tool calls rather than a block of generated prose, and Boiko et al. (2023) coupled such reasoning to liquid-handling hardware to design and execute reactions with minimal human intervention. That work sits atop a decade of laboratory automation — robotic flow-synthesis platforms, mobile robotic chemists that operate instruments designed for human hands, self-driving materials laboratories — whose intellectual lineage runs straight back to the closed loops of the symbolic era. On the software side, Lu et al. (2024) attempted the full loop — idea, code, experiment, paper — for machine-learning research, an experiment whose honest lesson concerns the limits of unsupervised generation at least as much as its promise. The surveys that consolidate the engineering patterns (Wang et al., 2023c; Ferrag et al., 2025) repay reading in sequence, because the vocabulary stabilised remarkably quickly.

Two forces make this shift urgent rather than merely interesting for working researchers. The first is volume: the literature has grown at a compounding rate for decades (Bornmann and Mutz, 2015), and no reading strategy scales with it. The second is adoption without method. A survey of 1,600 researchers found substantial and rapidly growing use of these tools alongside widespread anxiety about verification, provenance, and disclosure (Van Noorden and Perkel, 2023). Kitano (2021) frames the long-run target as an engine capable of a discovery worthy of a Nobel Prize; that framing is useful precisely because it is distant. The near-term target, and this book's target, is the delegation of the mechanical seventy per cent of a research week under conditions where the delegating scientist remains genuinely accountable for the result.

1.2 What Makes an Agent

The word “agent” has been stretched to cover a chatbot with a system prompt. For a book that will make quantitative claims about delegation, a tighter definition is needed. I will use the following. An *agent* is a system that, given a goal, maintains an observation history, selects actions from a bounded set of tools, executes them against an environment, observes the results, and repeats until a termination condition is met. The load-bearing words are *executes* and *observes*: a system that proposes a shell command is a generator, while a system that runs it and reads the exit code is an agent.

Formally, let g denote the goal specification, $\mathcal{T} = \{\tau_1, \dots, \tau_m\}$ the *tool registry* available to the agent, and h_t the *observation history* at step t , comprising the goal, all prior actions, and all prior observations. A *policy* π maps this state to an action:

$$a_t = \pi(g, h_t; \mathcal{T}), \quad a_t \in \mathcal{A}(\mathcal{T}) \cup \{\text{HALT}\}, \quad (1.1)$$

where $\mathcal{A}(\mathcal{T})$ is the set of well-formed invocations of tools in $\mathcal{T}$, including their arguments. The environment returns an observation $o_t = \text{Env}(a_t)$, and the history grows:

$$h_{t+1} = h_t \cup \{(a_t, o_t)\}. \quad (1.2)$$

Execution terminates when $a_t = \text{HALT}$, when a budget is exhausted, or when an *iteration cap* $t = K$ is reached. In RESEARCHOS the default cap on any self-correcting loop is $K = 5$, after which the loop escalates to a human rather than continuing to spend. That number is a judgement call, not a theorem; I chose it because in my own logs the marginal probability that a sixth attempt fixes what five attempts could not is small, while the marginal probability that the sixth attempt quietly redefines the task is not.

Equations (1.1) and (1.2) are deliberately austere, and their austerity is the point. Almost every design decision in this book is a decision about one of four objects: the goal specification g , the tool registry $\mathcal{T}$, the policy π , or the way h_t is compressed when it exceeds the context available. Prompting technique is a choice about π . Sandboxing is a constraint on $\mathcal{A}(\mathcal{T})$. Memory architecture, the subject of Section 1.4, is entirely about the representation of h_t . Human approval gates are interventions that suspend the loop between the selection of a_t and its execution.

Figure 1.2 unrolls the loop into the five stations that recur throughout this book: perceive, plan, act, observe, reflect. The *perceive-plan-act-observe-*

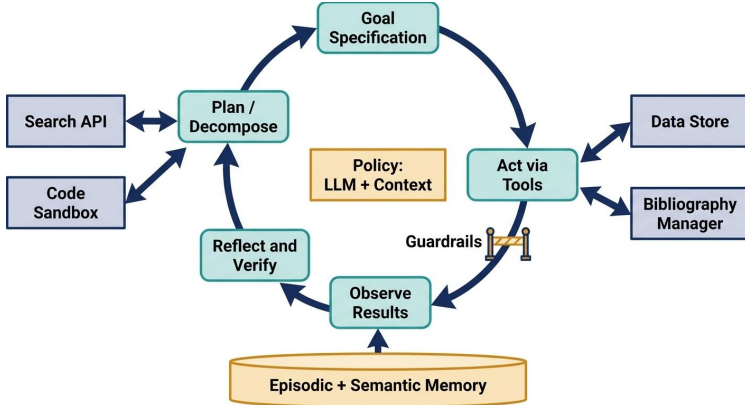


Figure 1.2: The agent loop and its five stations. Perception ingests the goal and the current history; planning decomposes it and selects the next action; action executes a tool call against a real environment; observation records the result, including failures; reflection updates the plan and, where warranted, the agent’s memory. The dashed return path marks the reflection step that distinguishes a self-correcting loop from a fixed pipeline, and the gate symbol marks where a human approval may suspend execution. Every traversal of the loop appends one record to the run ledger.

reflect loop is not a novel invention; it is the standard control loop of situated agents, restated with a language model in the policy slot. What language models contributed was a practical way to write π for open-ended, symbolically specified tasks without hand-coding a planner. Interleaving explicit reasoning traces with tool calls — the pattern popularised as ReAct — made the plan step legible and the act step conditional on it (Yao et al., 2022). Chain-of-thought prompting established that intermediate reasoning improves multi-step performance (Wei et al., 2022), and search over alternative reasoning structures extends the plan step from a single chain to a tree or a graph of candidate plans. Reflection can be implemented as verbal feedback carried across attempts (Shinn et al., 2023) or as iterative self-critique of one artifact, though in my experience critique grounded in tool output — a failing test, a compiler error, a schema violation — is far more reliable than ungrounded self-assessment, and I would not build a research loop that depends on the latter. For the underlying cognitive framing, Sumers et al. (2023) give the cleanest account of how these components compose.

Tool use deserves separate emphasis because it is where an agent touches reality. Models can be trained to invoke external interfaces at appropriate points (Schick et al., 2023), and the same machinery scales to registries of

thousands of endpoints, at which point correct interface selection and argument construction become the dominant error sources rather than reasoning as such. Delegating arithmetic to a program interpreter instead of performing it in text removes an entire class of error at a stroke. In research settings the most valuable tools are the boring ones: a bibliographic search connector over an open scholarly index (Priem et al., 2022), a sandboxed Python runtime, a job scheduler, and a diff-and-commit interface to a repository.

The contrast with a chat interface is now easy to state, and worth stating because the two are routinely conflated. In a chat, the human is the executor. The model proposes; you paste, run, read the traceback, and paste it back. Every observation passes through your attention, which means every error is caught at the moment it occurs and total work scales with the number of steps. In an agent, the loop closes without you. Steps become cheap — a 40-step debugging session that would cost two hours of paste-and-read costs eleven minutes of wall clock — but errors are no longer caught at the moment they occur. They are caught later, in aggregate, by verification. That single trade is the origin of almost everything difficult in this book.

1.3 Grading Autonomy: The L0–L5 Ladder

“Autonomous” is a binary word for a continuous quantity, and the resulting confusion is expensive: a vendor’s “fully autonomous literature review” and a research-integrity officer’s “autonomous” differ by roughly four rungs. I use the six-level ladder of Figure 1.3 throughout this book, and I recommend that groups adopting these systems label every workflow with a level in writing. The ladder grades three separable rights — who decides what to do, who executes it, and who verifies the result — because they can be assigned independently, and most real failures come from assigning them inconsistently.

Table 1.1 gives the operational definitions. L0 and L1 are familiar: a completion engine that suggests the next line, or an assistant that drafts a paragraph you then rewrite. At L2 the system executes multi-step plans but returns to the human between steps — the “chat with a code interpreter” regime. L3 is where the loop genuinely closes: the system plans, executes, and self-corrects within a scoped task, and the human verifies the final artifact. Most productive research use today sits at L3, and most of this book’s practical advice is about doing L3 well. At L4 the system additionally owns routine verification, escalating only anomalies; the human reviews a

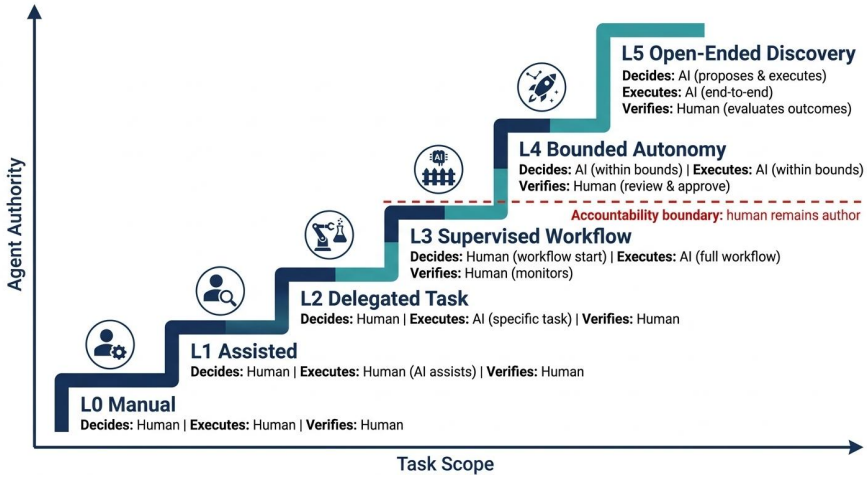


Figure 1.3: Six levels of delegated research autonomy. Moving up the ladder transfers decision, execution, and verification rights from the human to the system, in that order. The shaded band between L3 and L4 marks the accountability boundary: below it, a human has inspected every artifact that enters the scientific record; above it, the human has inspected a sample and delegated the remainder to automated verification. The cost of undetected failure rises sharply across that band, which is why crossing it is a governance decision rather than a configuration change.

sample plus the escalations. L5, in which the system sets its own research agenda, has no defensible place in a scientific workflow whose outputs carry a named author, and I include it in the ladder mainly so that the boundary is visible.

The right level is task-dependent, and the dependence runs not on task difficulty but on the cost structure of error. Reformatting 240 bibliography entries is an excellent L4 task: failures are visible, local, and cheap to repair, and an automated checker can compare each entry against the record it came from. Deciding which of 63 candidate studies meet an inclusion criterion is an L3 task even though it feels more mechanical, because a systematic misreading of the criterion propagates silently into every downstream conclusion and is invisible in the output itself. Interpreting a 0.4-point difference across five seeds is not delegable at any level above L1, because the judgement being made is the research. I have watched competent groups get this exactly backwards, pushing judgement-heavy tasks up the ladder because the agent's output reads authoritatively, while keeping mechanical tasks at L1 because those are the ones they know how to check.

Table 1.1: *The L0–L5 autonomy ladder. Rights over deciding, executing, and verifying are graded separately; the accountability boundary sits between L3 and L4, where routine verification passes from human to machine.*

L	Decides	Executes	Verifies	Typical research task	Cost of undetected failure
L0	Human	Human	Human	Autocomplete of a plotting call; spell check	Negligible
L1	Human	Human	Human	Drafting a related-work paragraph you then rewrite	Low: wasted minutes
L2	Human	Machine, step by step	Human	Interactive data cleaning with a code interpreter	Low: caught in session
L3	Human	Machine, closed loop	Human	Port a pipeline until its tests pass; screen 4,800 abstracts against stated criteria	Moderate: a wrong number enters a draft
L4	Human	Machine	Machine, sampled by human	Run a 60-run sweep, produce the statistics, flag anomalies	High: an unexamined claim reaches a reviewer
L5	Machine	Machine	Machine	Choose the research question and publish the answer	Severe: unattributable error in the record

Definition 1.1. Accountability boundary. The *accountability boundary* is the autonomy level at which routine verification of research artifacts passes from the human to the system. In the ladder of Table 1.1 it lies between L3 and L4. Below the boundary, every artifact that will enter the scientific record — every number, citation, figure, and claim — has been inspected by a named human before it does so. Above the boundary, the human has inspected a sample, and accountability for the remainder rests on the demonstrated error rate of an automated verifier together with the completeness of the provenance record. Crossing the boundary is therefore legitimate only when three conditions hold: the verifier’s false-negative rate has been measured on a labelled sample drawn from the same distribution as the work being checked; every artifact is reconstructible from the run ledger without re-executing the agent; and a named human accepts responsibility for the class of outputs as a whole.

The third condition is the one most often skipped, and it is the one that

makes the boundary meaningful rather than decorative. Responsibility does not become diffuse when work is delegated to software; it becomes concentrated in whoever signed the delegation. Tang et al. (2024) argue a version of the same point from the safety side for autonomous scientific systems specifically, and Kapoor et al. (2024) make the complementary methodological argument that claims of agent autonomy are routinely inflated by evaluations that measure neither cost nor reliability. Chapter 11 develops the governance machinery in full; here I want only to fix the vocabulary, because a group that cannot say “this workflow runs at L3 with G2 approval” will not be able to audit itself later.

1.4 Memory and Context

Equation (1.2) hides a practical impossibility. The history h_t grows without bound while the context window of any model is finite, so a real agent never conditions on h_t but on a projection $\phi(h_t)$ that must fit. The design of ϕ — what is kept verbatim, what is summarised, what is externalised to storage and retrieved on demand — is the single most consequential piece of agent engineering, and it is where naive implementations fail first. A 40-step debugging session generates on the order of 90,000 tokens of tool output; an agent that keeps the last twenty turns and drops the rest will, at step 33, cheerfully re-attempt the fix that already failed at step 6.

Figure 1.4 shows the four-store decomposition I recommend, and the allocation rule is simpler than the diagram suggests: material belongs in the store whose failure mode you can tolerate. *Working memory* is the live context — the current goal, the plan, the last few observations, the open file — and it is volatile by design; anything you would mind losing does not belong there. *Episodic memory* is the ordered record of what happened: which tool was called with which arguments, what came back, which attempt failed and how. It is append-only, it is what reflection reads, and in RESEARCHOS it is physically the same artifact as the run ledger. *Semantic memory* holds distilled and reusable knowledge: a one-paragraph method note about a paper, the observation that a particular cluster’s scheduler silently truncates array jobs above 1,000 tasks, an extracted claim with its supporting quotation. *Structured memory* holds typed records against a schema — the extraction table for a systematic review, the results table for a sweep, the bibliography — and it is the only store you can compute over reliably.

MEMORY ARCHITECTURE FOR LONG-RUNNING RESEARCH AGENT

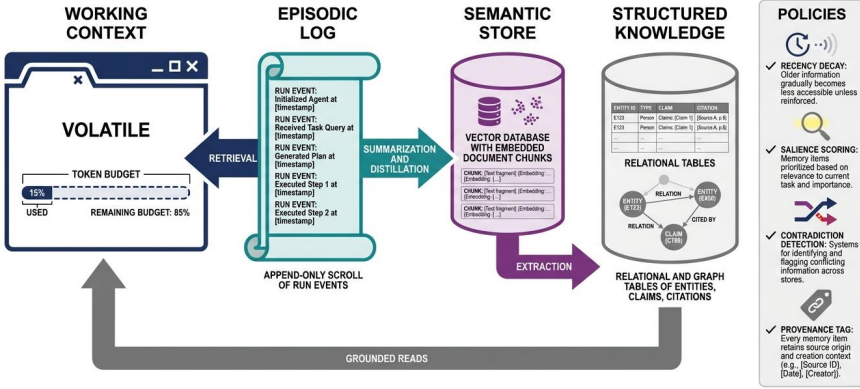


Figure 1.4: Working, episodic, semantic, and structured memory. Working memory is the live context window and is volatile by design. Episodic memory records what this agent did, in order, and is the substrate for reflection and for the run ledger. Semantic memory holds distilled, reusable knowledge — extracted claims, method notes, code idioms — indexed for retrieval. Structured memory holds typed records against a schema: extraction tables, run results, bibliographic entries. Distillation moves material rightward and loses detail; retrieval moves it leftward and must carry provenance with it, or the lost detail can never be recovered.

Retrieval binds the stores together. The pattern of conditioning generation on retrieved documents rather than on parameters alone (Lewis et al., 2020) is now standard, with dense vector indexes over passages as the default substrate. For agent memory specifically, Packer et al. (2023) treat the context window as a managed resource with explicit paging between fast and slow storage, and Zhang et al. (2024) survey the available mechanisms comparatively. Where the material to be retrieved is a corpus of interrelated claims rather than a set of independent passages — exactly the systematic-review case — graph-structured retrieval over extracted entities and relations answers global questions that flat top- k retrieval cannot, and Chapter 3 builds that index. Retrieval that critiques its own sufficiency before generating is a useful pattern wherever the cost of answering from insufficient evidence is high, which in research it usually is.

A practical *retrieval* policy for research agents should not score by semantic similarity alone. RESEARCHOS ranks candidate memory items m

against a query q by

$$s(m, q) = \alpha \cdot \text{sim}(e_m, e_q) + \beta \cdot \rho(m) + \gamma \cdot v(m) - \delta \cdot \lambda(m), \quad (1.3)$$

where $\text{sim}(\cdot, \cdot)$ is cosine similarity between embeddings, $\rho(m) \in [0, 1]$ is a provenance-strength score (a verbatim quotation with a DOI and a page anchor scores 1.0; an unsourced model-generated summary scores 0.2), $v(m)$ rewards recency for items whose truth decays — cluster quirks, interface behaviours, software defaults — and $\lambda(m)$ penalises items already contradicted by a higher-provenance item. The default weights in the reference deployment are $\alpha = 1.0$, $\beta = 0.6$, $\gamma = 0.15$, and $\delta = 1.2$. The large δ is deliberate: it is better to retrieve nothing than to retrieve a claim you have already refuted.

Distillation is the reverse operation, and it is where provenance is most often lost. When RESAGENT compresses forty tool observations into a three-sentence progress note, the note inherits the union of its sources’ identifiers rather than none of them; a summary whose sources cannot be enumerated is not written. *Contradiction detection* then makes the memory trustworthy over months rather than hours. Every time an item is written to semantic or structured memory, RESAGENT retrieves the $k = 8$ nearest existing items and asks a dedicated checker whether any of them assert incompatible propositions about the same entity. Detected contradictions are not resolved automatically. They are recorded as a linked pair, with both provenance tags, and surfaced to the human at the next approval gate — because in my experience roughly one contradiction in four is not an error at all but a genuine disagreement in the literature, and those are frequently the most interesting thing the agent found that week. Silent resolution destroys precisely the signal you want.

This brings us to the non-negotiable requirement. Every item in every store carries a *provenance tag* recording where it came from: a source identifier (DOI, commit hash, or file path with content hash), the extraction method and model version, the run in which it was produced, a timestamp, and a provenance-strength value. An untagged memory item is not knowledge; it is a rumour with good grammar. The requirement sounds bureaucratic until the first time a reviewer asks where a number came from and you can answer in nine seconds. Chapter 10 makes the case in full, and Chapter 3 shows what the tags look like for extracted claims.

1.5 The RESEARCHOS Reference Architecture

The remainder of this book is organised around a single reference architecture, RESEARCHOS, and a single reference deployment of it, RESAGENT. The architecture is not a product and there is nothing proprietary about it; it is a way of arranging components so that each chapter can develop one part without re-explaining the whole. Figure 1.5 shows the five layers.

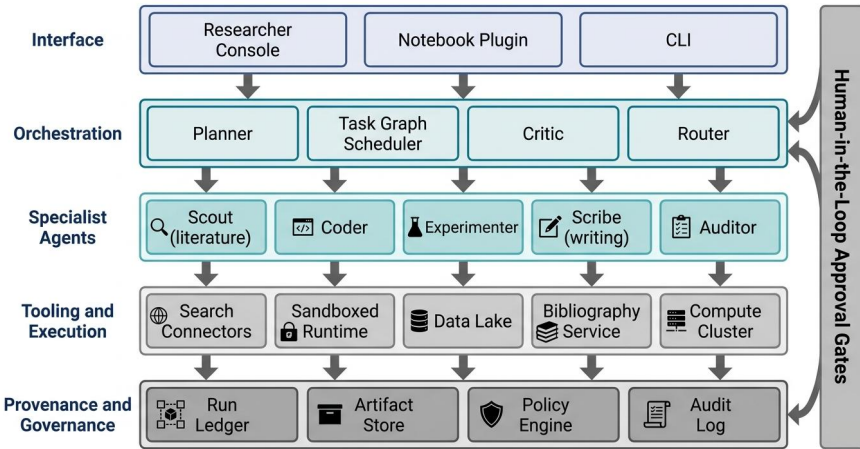


Figure 1.5: Interface, Orchestration, Specialist Agents, Tooling, and Provenance and Governance. The *Interface* layer is how a human enters and leaves the system. *Orchestration* decomposes goals into a task graph, schedules it, routes each task to a specialist, and critiques the result. The five specialist agents — Scout, Coder, Experimenter, Scribe, Auditor — hold role-specific policies, tools, and memory scopes. *Tooling* provides the only sanctioned paths to the outside world. The *Provenance and Governance* layer spans all four others: every action is mediated by the Policy Engine before execution and recorded in the Run Ledger after it.

The *Interface* layer is deliberately thin: a console, a notebook plugin, and a command-line entry point that all construct the same object, a *task contract*. The *Orchestration* layer contains a Planner that decomposes a goal into a directed acyclic *task graph*, a Scheduler that executes ready nodes subject to resource and budget constraints, a Router that assigns each node to a specialist, and a Critic that evaluates node outputs against the contract’s acceptance criteria before results propagate. Chapter 6 develops the task graph in detail; the multi-agent conversation frameworks in the literature, of which Wu et al. (2023) is representative, solve the same coordination

problem with a different topology, substituting message passing between conversational roles for an explicit graph with a scheduler.

The *Specialist Agents* layer holds five roles, and each has a chapter. **Scout** searches, screens, and de-duplicates literature, and is developed in Chapter 2, with the synthesis and claim-extraction side in Chapter 3. **Coder** writes, refactors, and tests research software (Chapter 4) and diagnoses failures (Chapter 5). **Experimenter** designs and executes computational experiments (Chapter 6), and its statistical duties are the subject of Chapter 7. **Scribe** drafts, revises, and formats manuscripts (Chapter 8) and prepares responses to reviewers (Chapter 9). **Auditor** verifies — statistics, citations, numeric consistency between text and results files, and policy compliance — and appears in every chapter, with its verification machinery consolidated in Chapters 9 and 10 and its adversarial duties in Chapter 11. Chapter 12 runs all five end to end on a single study.

The *Tooling* layer provides Search Connectors, a Sandboxed Runtime, a Data Lake, a Bibliography Service, and a Compute Cluster interface. Its defining property is exclusivity: agents have no other route to the world. The *sandbox* is a rootless container with a read-only base image, a seccomp profile, an egress allow-list, and default caps of 4 CPU, 8 GB of memory, and 900 seconds of wall clock, with token and USD budget ceilings enforced by a *circuit breaker*. Chapter 11 explains why each of those exists, and why an allow-list rather than a deny-list is the only defensible choice once you accept that retrieved documents are untrusted input (Greshake et al., 2023).

The fifth layer — *Provenance and Governance*, comprising the *Run Ledger*, *Artifact Store*, *Policy Engine*, and *Audit Log* — is drawn as a band spanning the other four because that is what it is. Provenance is a layer and not a practice, and the distinction carries the most important architectural claim in this chapter. A practice is something conscientious people do: it degrades under deadline pressure, it varies between group members, and it fails silently. A layer is something the system cannot operate without. In RESEARCHOS a tool call that cannot be recorded does not execute, because the ledger write sits on the critical path of the call rather than being appended afterwards by a well-meaning wrapper. The provenance literature reached this conclusion for scientific workflows two decades ago (Freire et al., 2008) and went on to standardise an interchange model for it, while the community’s data-stewardship principles assume machine-actionable metadata rather than human diligence (Wilkinson et al., 2016). Agentic

systems raise the stakes because they generate artifacts faster than any human can annotate them.

Ledger entries are append-only JSON Lines records. Each carries exactly these fields:

- `run_id`, `parent_run_id` — identity and lineage;
- `agent`, `agent_version` — which policy produced the record;
- `tool_calls` — the ordered invocations, with arguments;
- `inputs_hash`, `outputs_hash` — content addresses;
- `wall_clock_s`, `token_cost`, `usd_cost` — resources;
- `human_approvals`, `policy_decisions` — governance events.

Listing 1.1 gives the append function; note that it flushes and syncs before returning, so a crash mid-task leaves a truncated ledger rather than a missing one. Listing 1.2 is a minimal but complete agent loop with a tool registry and an iteration cap, and Listing 1.3 shows the authority declaration that the Policy Engine consults before every tool call.

```

1 import json, hashlib, os, time
2 from pathlib import Path
3
4 LEDGER = Path(os.environ.get("RESEARCHOS_LEDGER", "ledger.jsonl"))
5
6 def _hash(obj) -> str:
7     blob = json.dumps(obj, sort_keys=True, default=str).encode()
8     return "sha256:" + hashlib.sha256(blob).hexdigest()[:32]
9
10 def append_ledger(run_id, parent_run_id, agent, agent_version,
11                  tool_calls, inputs, outputs, wall_clock_s,
12                  token_cost, usd_cost, human_approvals=(),
13                  policy_decisions=()):
14     """Append one immutable step record. Raises on failure so the
15     caller aborts rather than proceeding unrecorded."""
16     record = {
17         "ts": time.strftime("%Y-%m-%dT%H:%M:%SZ", time.gmtime()),
18         "run_id": run_id,
19         "parent_run_id": parent_run_id,
20         "agent": agent,
21         "agent_version": agent_version,
22         "tool_calls": list(tool_calls),
23         "inputs_hash": _hash(inputs),
24         "outputs_hash": _hash(outputs),
25         "wall_clock_s": round(wall_clock_s, 3),
26         "token_cost": int(token_cost),

```

```

27         "usd_cost": round(usd_cost, 4),
28         "human_approvals": list(human_approvals),
29         "policy_decisions": list(policy_decisions),
30     }
31     with LEDGER.open("a", encoding="utf-8") as fh:
32         fh.write(json.dumps(record, ensure_ascii=False) + "\n")
33         fh.flush()
34         os.fsync(fh.fileno())
35     return record["outputs_hash"]

```

Listing 1.1: *Ledger append: one canonical JSON Lines record per agent step, written on the critical path.*

```

1 class ToolRegistry:
2     def __init__(self):
3         self._tools = {}
4
5     def register(self, name, fn, schema):
6         self._tools[name] = (fn, schema)
7
8     def schemas(self):
9         return {n: s for n, (_, s) in self._tools.items()}
10
11     def call(self, name, args):
12         if name not in self._tools:
13             return {"ok": False, "error": "unknown tool " + name}
14         fn, _ = self._tools[name]
15         try:
16             return {"ok": True, "result": fn(**args)}
17         except Exception as exc: # observed, never raised
18             return {"ok": False,
19                     "error": type(exc).__name__ + ": " + str(exc)}
20
21     def run_agent(goal, policy, registry, run_id, max_iters=5):
22         history = [{"role": "goal", "content": goal}]
23         for step in range(max_iters):
24             action = policy.next_action(goal, history, registry.schemas()
25             ↪ )
26             if action["type"] == "halt":
27                 return {"status": "done", "answer": action["answer"],
28                         "steps": step}
29             obs = registry.call(action["tool"], action["args"])
30             history.append({"action": action, "observation": obs})
31             append_ledger(run_id, None, policy.name, policy.version,
32                           [action], goal, obs, action.get("secs", 0.0),
33                           action.get("tokens", 0), action.get("usd", 0.0)
34             ↪ )
35             if not obs["ok"]:
36                 history.append(policy.reflect(history))
37         return {"status": "escalate", "reason": "iteration cap reached",
38                 "history": history, "steps": max_iters}

```

Listing 1.2: *A minimal agent loop: tool registry, iteration cap, and escalation instead of unbounded retry.*

```

1 # policy/scout.authority.yaml
2 agent: scout
3 agent_version: "2.3.1"
4 autonomy_level: L3
5 allow:
6   - tool: search.openalex
7     rate_limit_per_min: 30
8   - tool: fetch.pdf
9     egress_allowlist: [api.openalex.org, api.semanticscholar.org]
10  - tool: memory.write
11    stores: [episodic, semantic, structured]
12 deny:
13   - tool: shell.exec
14   - tool: memory.delete
15   - tool: bibliography.publish
16 budgets:
17   usd_ceiling: 25.00
18   token_ceiling: 4000000
19   wall_clock_s: 3600
20 escalate_to_human_when:
21   - iteration_cap_reached
22   - contradiction_detected_in_semantic_memory
23   - budget_fraction_used_above: 0.8
24 requires_approval_gate: G1

```

Listing 1.3: Authority declaration consulted by the Policy Engine before every tool call (YAML syntax).

Case Study: ResAgent — the meta-study and gate G1

RESAGENT is the reference deployment of RESEARCHOS used throughout this book, and the scientific problem it works on is deliberately methodological rather than substantive: *does the choice of learning-rate schedule materially change reported accuracy in published image-classification benchmarks, and is the reported difference distinguishable from seed variance?* The question makes a good running example for four reasons. It requires a real literature pass, so Scout has honest work. It requires re-implementation, so Coder does too. It has a bounded experimental core — 3 schedules $\times$ 5 seeds $\times$ 4 datasets = 60 runs, roughly 6 GPU-hours on the reference cluster — so Experimenter’s sweep is affordable. And the answer might well be “no”, which keeps everyone honest.

The first gate is **G1, scope approval**, and it comes before any search executes. RESAGENT converts the researcher’s one-sentence question into the task contract of Listing 1.4 and presents it for signature. What the researcher actually approves at G1 is not a plan of work but a set

of commitments that will be expensive to change later: the exact population of studies in scope (peer-reviewed image-classification papers, 2015–2025, reporting top-1 accuracy on at least one of four named datasets, with the schedule stated or inferable from released code); the four named sources and the seed query set; the inclusion and exclusion criteria in testable form; the definition of the primary outcome; the maximum spend; and the autonomy level (L3, meaning every included and excluded record is auditable and the final corpus is human-reviewed). Approving G1 took the author 22 minutes on the first pass, most of it spent rejecting the agent’s initial inclusion criterion, which would have admitted preprints without released code and thereby made the schedule field unverifiable for perhaps a third of the corpus.

The artifact produced by G1 is a signed scope document: the frozen contract, the approver’s identity, a UTC timestamp, a content hash of the contract, and the run identifier `run-0417-scope-01`, which becomes the `parent_run_id` of every downstream Scout, Coder, Experimenter, Auditor, and Scribe run in the study. Gate G2 — the analysis plan and pre-registration freeze — is taken up in Chapter 6; G3 and G4 in Chapters 7 and 8.

```

1 # contracts/meta_study_scope.yaml
2 contract_id: meta-lr-schedule-scope
3 run_id: run-0417-scope-01
4 autonomy_level: L3
5 gate: G1
6 question: >
7   Does the choice of learning-rate schedule materially change
8   reported top-1 accuracy in published image-classification
9   benchmarks, and is any reported difference distinguishable
10  from seed variance?
11 population:
12   venues: [peer_reviewed, preprint_with_code]
13   years: [2015, 2025]
14   datasets: [CIFAR-10, CIFAR-100, ImageNet-1k, SVHN]
15 sources: [openalex, semantic_scholar, dblp, arxiv]
16 inclusion:
17   - reports_top1_accuracy_on_named_dataset
18   - schedule_stated_or_inferable_from_released_code
19   - reports_seed_count_or_release_permits_reruns
20 exclusion:
21   - survey_or_position_paper
22   - no_accuracy_table
23 primary_outcome: delta_top1_between_schedules_same_backbone
24 deliverables:

```

```

25 - screened_corpus.jsonl
26 - extraction_table.csv
27 - scope_report.md
28 acceptance_criteria:
29   - every_record_has_provenance_tag
30   - inclusion_decisions_reproducible_from_ledger
31   - sampled_precision_on_40_records_at_least: 0.90
32 budget: {usd_max: 30.00, wall_clock_s: 5400}
33 verification: {method: stratified_sample, n: 40, verifier: human}
34 approvals_required: [G1]

```

Listing 1.4: *The G1 task contract for the meta-study scoping task (YAML syntax).*

1.6 The Delegation Contract and an Economics of Verification

Delegation is an exchange: you give up the direct knowledge that comes from doing the work, and you receive time. Whether the exchange is favourable is an arithmetic question, and it is worth actually doing the arithmetic, because the answer determines what the rest of this book has to be about.

Let c_h be the cost of doing a task yourself, measured in units of your own attention — I will use researcher-hours, converting money to hours at a fully-loaded rate. Delegating the same task incurs an agent cost c_a (tokens, compute, and the time you spend writing the contract), a *verification cost* c_v (the time you spend establishing that the output is correct), and, with probability p_f , a failure requiring repair at cost c_r followed by re-verification at cost c'_v . The expected cost of delegation is

$$\mathbb{E}[C_{\text{del}}] = c_a + c_v + p_f (c_r + c'_v), \quad (1.4)$$

and delegation pays exactly when

$$c_a + c_v + p_f (c_r + c'_v) < c_h. \quad (1.5)$$

Rearranging gives the break-even failure probability, which is the most useful diagnostic form:

$$p_f^* = \frac{c_h - c_a - c_v}{c_r + c'_v}, \quad (1.6)$$

with delegation favourable when $p_f < p_f^*$ and never favourable — whatever the agent's reliability — when $c_a + c_v \geq c_h$.

Take the G1 scoping task of Section 1.5 with illustrative numbers from the author's own logs. Done by hand, constructing the query set, screening the returns, and writing a defensible scope document costs $c_h = 6.0$ hours.

Delegated to Scout at L3, the run costs about USD 18 in tokens and compute, which at USD 60 per researcher-hour is 0.30 h, plus 0.25 h to write and review the contract, so $c_a = 0.55$ h. Verification means reading the scope report, checking a stratified sample of 40 inclusion decisions against their provenance tags, and re-running two queries by hand: $c_v = 1.10$ h. The observed failure rate on this class of task in the reference deployment is $p_f = 0.25$; repair is usually a criterion clarification and a re-run, $c_r = 0.35$ h, followed by a lighter second check, $c'_v = 0.80$ h. Then $\mathbb{E}[C_{\text{del}}] = 0.55 + 1.10 + 0.25(0.35 + 0.80) = 1.94$ h against $c_h = 6.0$ h: a saving of 4.06 h, an amplification of $3.1\times$, and from Equation (1.6) a break-even rate of $p_f^* = (6.0 - 0.55 - 1.10)/1.15 = 3.78$. Since that exceeds 1, delegation pays at *any* failure rate whatsoever. This is the comfortable case.

Now break one number and change nothing else. Suppose the output carries no provenance tags, so that verifying an inclusion decision means re-finding the paper, locating the accuracy table, and confirming the schedule by hand. Verification rises to $c_v = 4.50$ h and $c'_v = 4.00$ h. Nothing about the agent has changed — same model, same tools, same 25 per cent failure rate — yet $\mathbb{E}[C_{\text{del}}] = 0.55 + 4.50 + 0.25(0.35 + 4.00) = 6.14$ h, and delegation now costs more than doing the work yourself. The break-even reliability has collapsed from “anything” to $p_f^* = 0.22$. This is not a contrived example; it is the single most common way that agent adoption fails in research groups, and it fails invisibly, because the group measures the agent’s speed and never measures its own verification burden.

The general form of the point falls out of Equation (1.4) by taking the agent to be free, which is roughly the direction hardware and model prices have been moving:

$$\lim_{c_a \rightarrow 0} \mathbb{E}[C_{\text{del}}] = c_v + p_f(c_r + c'_v) \geq c_v, \quad A_{\text{max}} = \frac{c_h}{c_v}, \quad (1.7)$$

where A_{max} is the maximum achievable amplification of a researcher’s effort. Verification cost, not agent capability, sets the ceiling. In the comfortable case above the ceiling is $6.0/1.10 = 5.5\times$; in the broken case it is $1.3\times$. No improvement in the underlying model moves either number.

Key Idea: Verification cost is the binding constraint

Delegation to an agent is profitable only when checking the result is cheap relative to producing it. Because the expected cost of delegation is bounded below by the verification cost, the maximum amplification of research effort is $A_{\max} = c_h/c_v$ regardless of how capable the agent becomes. It follows that the highest-return engineering work in agentic research is not making agents more capable but making their outputs cheaper to verify: provenance tags on every claim, deterministic re-execution, tests that encode acceptance criteria, structured outputs you can compute over, and ledgers that answer “where did this number come from?” in seconds rather than hours. Most of the remaining chapters of this book are, read in this light, chapters about lowering c_v .

The *delegation contract* is the instrument that operationalises this. A task handed to an agent should specify, before execution, what the deliverable is; what acceptance criteria the Critic and Auditor will apply; which tools and data are in scope; the budget and wall-clock ceilings; the autonomy level; the approval gates; and — the field most often omitted — how the output will be verified and by whom. Listing 1.4 is such a contract, and its acceptance-criteria block exists precisely to make c_v small. A requirement that every record carry a provenance tag converts verification from re-doing the work into sampling it, and a stated sampled-precision threshold of 0.90 on 40 records converts a vague judgement into a thirty-minute procedure with a number at the end. Writing the contract costs fifteen minutes. In the accounting above, it is those fifteen minutes that make the other 4.06 hours available.

In Practice: measure your own c_v before scaling up

Before adopting an agentic workflow across a group, run the task twice by hand and twice delegated, and time all four. You need four numbers, not an impression: c_h from the manual runs, c_a from the ledger’s `usd_cost` and `wall_clock_s` fields, c_v from a stopwatch on your own verification, and p_f from how many of the delegated runs needed repair. Then compute $A_{\max} = c_h/c_v$. If it is below about $2\times$, do not scale the workflow; instead spend the next iteration reducing c_v — add provenance tags, add acceptance tests, make the output a table instead

of prose — and re-measure. Groups that scale before measuring end up with a verification backlog, which is the agentic equivalent of technical debt and accrues interest in the same way.

Two boundary conditions deserve mention. Delegation also has value this model omits: an agent that screens 4,800 records performs a task you would never have attempted by hand, so c_h is not six hours but effectively infinite, and any finite $\mathbb{E}[C_{\text{del}}]$ wins. Conversely, the model omits a real cost — the erosion of tacit knowledge that comes from doing the work. A student who never once ports a data loader by hand will not recognise the smell of a broken one. Messeri and Crockett (2024) argue that AI-mediated research can produce illusions of understanding at the level of an entire field, and I take the concern seriously enough to recommend that early-career researchers deliberately keep some first-bin work undelegated, chosen for what it teaches rather than for what it costs.

1.7 What Agents Are Currently Bad At

A practitioner’s guide owes the reader an honest account of failure. What follows is drawn from published evaluations and from the reference deployment’s own logs, and it is organised by failure mode rather than by severity, because the modes have different detection strategies and conflating them is itself a failure mode.

The most discussed is *fabrication*: confident assertion of things that are not so. In research settings its costliest form is *miscitation* — a real paper cited for a claim it does not make, or a plausible-looking reference that does not exist at all. The general phenomenon and its taxonomy are surveyed for large language models by Huang et al. (2023), and the survey’s most useful lesson for practitioners is that fabrication is not a single defect with a single cure. The mitigation that works is not exhortation but architecture: require every claim to carry a resolvable identifier and a verbatim supporting span, then check the span mechanically (Gao et al., 2023a). Chapter 3 builds that checker. *Plan drift* is subtler. An agent thirty steps into a task is conditioning on a compressed history, and compression loses constraints before it loses momentum; the system continues working competently on a task that is no longer the one it was given. Detection requires comparing the current

working objective against the frozen contract at intervals, which is one reason the contract is a file rather than a conversation.

Scope creep is plan drift's more dangerous cousin, because its outputs look better rather than worse. Asked to fix a failing test, an agent refactors three modules, updates a dependency, and reformats a file; every change is individually defensible and the diff is now unreviewable. *Tool misuse* covers well-formed calls with wrong semantics — the correct interface invoked on the wrong column, a statistical test applied to the wrong grouping — and it is the mode least visible in output text, since a wrong number and a right number look identical. *Overconfident under-specification* is the failure to ask: given an ambiguous instruction, current systems overwhelmingly guess and proceed rather than stopping to clarify, and the guess is not flagged as one. *Reward hacking* is optimisation of the stated metric at the expense of the intended one; an agent told to maximise a validation score will, given tools, eventually find the validation set. Amodei et al. (2016) laid out the general shape of this problem years before agents made it a daily operational concern. Finally, *nondeterminism*: sampled decoding, variable tool latencies, and a well-documented sensitivity to incidental details of prompt formatting together mean that the same task run twice may take different paths, which breaks the mental model of debugging that most researchers bring from deterministic software.

The evaluation literature supplies the quantitative discipline that anecdote cannot. General assistant benchmarks show that tasks trivial for humans remain hard for agents whenever they require several dependent tool calls (Mialon et al., 2023), and interactive web and desktop environments expose the brittleness of long action sequences in a way that single-turn evaluations cannot. On real software-engineering issues, resolution rates began in the low single digits and rose substantially once the agent's interface to the repository was engineered rather than improvised (Jimenez et al., 2023). Read alongside Kapoor et al. (2024), whose central methodological complaint is that reported agent performance is frequently uncontrolled for cost and unstable across reruns, the honest summary is this: capability is real, improving, and systematically overstated by demonstrations. Chapter 12 asks you to measure your own pipeline rather than trust anyone's published numbers, including mine.

Table 1.2: *A failure taxonomy for research agents, with the detection strategy that works in practice and the chapter that develops it.*

Failure mode	Observable signature	Detection that works	Chapter
Fabrication, miscitation	Fluent claim with no resolvable source; reference resolves to unrelated work	Mechanical source matching on every claim	span-to- 3, 8
Plan drift	Late steps optimise a goal absent from the contract	Periodic objective statement, diffed against frozen contract	re- 6
Silent scope creep	Diff far larger than the task warrants	Change budget; diffs touching unlisted files	reject 4
Tool misuse	Well-formed call, wrong semantics; plausible wrong number	Assertions on units, shapes, and group counts; golden tests	5
Overconfident under-specification	Ambiguity resolved silently by guessing	Require an explicit assumptions block; reward clarifying questions	as- 4
Reward hacking a metric	Metric improves while the artifact degrades	Held-out metric the agent cannot access; adversarial audit	7
Nondeterminism	Two identical runs, two different paths	Seed and version pinning; n -of- m agreement on decisions	10

Caution: fluency is not evidence

The output of a competent research agent is well organised, correctly formatted, appropriately hedged, and pleasant to read. None of those properties correlates with correctness, and all of them suppress the reader’s scepticism. A hand-made draft signals its own uncertainty through rough edges; an agent-made draft does not. Assume, therefore, that your intuitive error-detection reflexes are miscalibrated on this material, and replace them with procedures: sample and check a fixed number of items rather than reading for plausibility; verify every number against the file it came from rather than against your memory of the experiment; and treat any claim without a resolvable provenance tag as absent rather than merely as unsupported. The most expensive mistakes I have seen were not made by agents. They were made by researchers who read good prose and inferred good work.

Remark 1.1. A useful corollary of Section 1.6 is that the failure modes in Table 1.2 are costly in proportion to how much they inflate c_v rather than to how often they occur. Nondeterminism has a high incidence and a modest cost, because pinning seeds and versions largely contains it. Miscitation has a lower incidence and a far higher cost, because detecting it without mechanical span checking requires re-reading the sources — which is the whole task again. Rank your mitigations by their effect on c_v , not by the frequency of the underlying error.

Chapter Summary

Machine assistance in science has passed through three regimes, and the current one differs in kind rather than in degree. Symbolic systems inferred over encoded knowledge and handed juxtapositions to humans. Statistical systems learned functions accurate enough to serve as observation channels. Agentic systems select and execute actions and observe their consequences. That last capability is what makes them useful and what makes them dangerous, and it is why the governing formalism of this book is a loop — $a_t = \pi(g, h_t; \mathcal{T})$ with $h_{t+1} = h_t \cup \{(a_t, o_t)\}$ — rather than a mapping from prompt to text. Everything else is a decision about the goal, the tools, the policy, or the representation of the history.

Because autonomy is continuous, it needs grading. The L0–L5 ladder separates the rights to decide, to execute, and to verify, and locates the accountability boundary between L3 and L4, where routine verification passes from human to machine. Crossing that boundary is a governance act with three preconditions: a measured verifier error rate, artifacts reconstructible from the ledger, and a named human who accepts responsibility for the class of outputs. Most productive research work today belongs at L3, and the appropriate level depends on the cost structure of error rather than on the difficulty of the task — which is why bibliography reformatting can safely sit above screening decisions that feel far more mechanical.

The architecture that carries the rest of the book has five layers and five specialist agents. Interface, Orchestration, Specialist Agents, and Tooling are conventional; the fifth layer, Provenance and Governance, spans the others because provenance implemented as a practice degrades under deadline pressure while provenance implemented as a layer does not. In RESEARCHOS, a tool call that cannot be recorded does not execute. Scout, Coder, Experimenter, Scribe, and Auditor each receive a chapter or two,

and RESAGENT works throughout on a single methodological meta-study whose scope was frozen at gate G1 into a signed document carrying the run identifier `run-0417-scope-01`.

The economics justify that emphasis. Delegation pays when $c_a + c_v + p_f(c_r + c'_v) < c_h$, and since the expected cost of delegation is bounded below by the verification cost, achievable amplification is capped at $A_{\max} = c_h/c_v$. In the worked example, stripping provenance from an otherwise identical agent run moved the same task from a threefold saving to a net loss. Capability improvements cannot repair that; only cheaper verification can. Set against this are seven recurring failure modes — fabrication and miscitation, plan drift, silent scope creep, tool misuse, overconfident underspecification, reward hacking, and nondeterminism — each with a detection strategy that must be procedural rather than intuitive, because fluent output disarms intuition.

Chapter 2 takes up the first specialist in earnest. It develops Scout: how to turn a research question into a reproducible search strategy across open scholarly indexes, how to screen thousands of records against testable inclusion criteria while keeping every decision auditable, how to measure screening precision and recall against a human-labelled sample, and how the reporting conventions of systematic review map onto an agentic pipeline. The meta-study contract signed at G1 in this chapter is its input.

Exercises and Discussion Questions

1. Keep an honest time log for one working week, then classify every hour as delegable or non-delegable using the criterion of Section 1.1: can the success criteria be written down in advance? Report the ratio and identify the three largest delegable blocks.
2. Explain, using Equations (1.1) and (1.2), why a system that proposes shell commands for a human to run is not an agent under this book's definition. Then argue the opposite position as strongly as you can, and say which argument you find more useful for engineering purposes.
3. Assign an autonomy level from Table 1.1 to each of the following: writing a data-loading script; deciding whether a preprint meets an inclusion criterion; regenerating fourteen figures to journal specification; choosing which of three competing explanations of an anomaly

- to pursue. Justify each assignment by the cost structure of its errors, not by its difficulty.
4. Compute p_f^* from Equation (1.6) for a task with $c_h = 3.0$ h, $c_a = 0.4$ h, $c_v = 1.6$ h, $c_r = 0.5$ h, and $c'_v = 1.2$ h. Then find the value of c_v at which delegation stops paying for an agent with $p_f = 0.3$, holding everything else fixed, and report the corresponding $A_{\max}$.
 5. Implement Listing 1.2 against two real tools — a literature search connector and a Python execution sandbox — and run it on the question “how many of the last five best papers at a venue of your choice released training code?”. Log every step with Listing 1.1 and report total token cost, wall clock, and the number of iterations to halt.
 6. Extend the ledger record with a `schema_version` field and write a migration that upgrades existing JSON Lines records without violating the append-only guarantee. State precisely what “without violating” must mean for the guarantee to survive a migration.
 7. Design a contradiction-detection test set: construct ten pairs of memory items, five genuinely incompatible and five superficially similar but compatible, then measure your detector’s precision and recall. Report which compatible pairs it flagged and diagnose why.
 8. Critique the accountability boundary of Section 1.3. Suppose a published paper contains an error introduced by an L4 verification step whose false-negative rate had been correctly measured and disclosed. Who is responsible: the delegating researcher, the group leader who approved L4 operation, the tool’s authors, or nobody? Defend a position, and state what a journal’s disclosure policy would have to require to make your position enforceable.

Further Reading

For the historical arc, begin with Swanson (1986), which is short, beautifully argued, and establishes that machine-surfaced juxtaposition plus human judgement can yield a testable hypothesis — the epistemic template this book still follows. Then read King et al. (2009) for the first genuinely closed scientific loop, and Kitano (2021) for the most ambitious statement of where such loops might lead; taken in that order they make clear how little of the

agenda is new and how much of the engineering is. Wang et al. (2023b) is the best single survey of the statistical era's scientific impact, and Jumper et al. (2021) remains the clearest case of a predictor becoming an instrument. Boiko et al. (2023) is the closest thing to a published existence proof for the agentic regime in a wet-lab setting.

On agents as engineering, Yao et al. (2022) is the paper to read first, because it makes the plan-act interleaving explicit in a form you can implement in an afternoon, and Sumers et al. (2023) is the paper to read second, because it supplies the vocabulary for everything that follows. Wang et al. (2023c) and the more recent Ferrag et al. (2025) cover the design space at survey depth; the peer-reviewed version of the former appeared in *Frontiers of Computer Science* (Wang et al., 2024c), and that is the version to cite in a paper. de Lamo Castrillo et al. (2025) is a useful consolidated treatment for readers who want construction details, Durante et al. (2024) widens the frame to multimodal and embodied interaction, which matters once your instruments emit images or video rather than tables, and Chowa et al. (2025) reviews tool use specifically and is the fastest way to check whether a pattern you are about to invent already has a name. For memory specifically, Packer et al. (2023) reframes context as a managed resource and Zhang et al. (2024) compares mechanisms; pair them with Lewis et al. (2020) for the retrieval foundation on which both depend.

For calibration — which matters more than capability in a research setting — read Kapoor et al. (2024) carefully and more than once; its argument that agent evaluations routinely omit cost and reliability is the antidote to every demonstration video you will be sent. Mialon et al. (2023) shows how far current systems remain from competence on tasks humans find easy. Bubeck et al. (2023) belongs on the same shelf for the opposite reason: read it sceptically and notice that its evidence is curated transcripts rather than measured task success, which is exactly the distinction you will need when a demonstration lands in your inbox. Then Huang et al. (2023) maps the hallucination problem you will spend the most time defending against. On the human side, Van Noorden and Perkel (2023) documents what researchers are actually doing with these tools, and Messeri and Crockett (2024) makes the strongest available case that the deeper risk is not error but the illusion of understanding. Read that last one before you delegate anything you care about.