CODANICS

Python ka Chilla · Build AI Agents

A Complete A–Z Product Development Roadmap

FIVE AI RESEARCH AGENTS

Building the AI Research OS

guides · Python code · market · learning path

1 Research Discovery Agent

2 Literature Review Agent

3 Research Experiment Agent

4 Writing & Publication Agent

5 PhD Supervisor Agent

Dr. Muhammad Aammar Tufail

Founder/CEO, Codanics

codanics.com

YouTube @codanics · Python ka Chilla: Build AI Agents

Five AI Research Agents

Building the AI Research OS — A Complete A–Z Product Development Roadmap

Dr. Muhammad Aammar Tufail

2026-09-22

Table of contents

1 Five AI Research Agents **6**
- 1.1 The five agents . . . 8
- 1.2 Who this book is for . . . 8
- 1.3 How the book is organised . . . 8
- 1.4 How to run the code . . . 8
- 1.5 The design philosophy . . . 9
- 1.6 Codanics links . . . 9

I Part I — The Council & the Opportunity **10**

2 The Council of Five — Deliberation & Decision **11**
- 2.1 The council members . . . 11
- 2.2 The question put to the council . . . 11
- 2.3 Round 1 — Each member's opening position . . . 11
- 2.4 Round 2 — Debate & resolutions . . . 12
- 2.5 The decision . . . 13
- 2.6 Council sign-off . . . 14

3 Guide 0 — The Opportunity: AI Research Agents & the "AI Research OS" **16**
- 3.1 Why research is the best vertical for AI agents . . . 16
- 3.2 Market size (order-of-magnitude, for planning) . . . 16
- 3.3 The five agents at a glance . . . 17
- 3.4 Ranking (from the council) . . . 17
- 3.5 The AI Research OS (Phase 2 vision) . . . 17
- 3.6 Common architecture (all five agents share it) . . . 18
- 3.7 Where the money is (summary) . . . 18

II Part II — Guides: one per agent **20**

4 Guide 1 — AI Research Discovery Agent **21**
- 4.1 "Your AI Research Scientist" . . . 21
- 4.2 The problem . . . 21
- 4.3 The solution . . . 21
- 4.4 Example . . . 21
- 4.5 Architecture . . . 22
- 4.6 Build steps (what the code does) . . . 22
- 4.7 Extending it (product features) . . . 22
- 4.8 Market value . . . 23
- 4.9 How to sell it . . . 23
- 4.10 Risks & mitigations . . . 23
- 4.11 90-day roadmap . . . 23

5 Guide 2 — AI Literature Review Agent **24**
- 5.1 "Autonomous Systematic Review Scientist" . . . 24
- 5.2 The problem . . . 24
- 5.3 The solution . . . 24
- 5.4 Architecture . . . 25
- 5.5 Build steps . . . 25
- 5.6 Product extensions . . . 25

5.7 Market value . . . 25
5.8 How to sell it . . . 26
5.9 Risks & mitigations . . . 26
5.10 90-day roadmap . . . 26

6 Guide 3 — AI Research Experiment Agent 27
6.1 "Virtual Research Assistant" . . . 27
6.2 The problem . . . 27
6.3 The solution . . . 27
6.4 Architecture — *code does the math, the LLM does the reasoning* . . . 28
6.5 Build steps . . . 28
6.6 Product extensions . . . 28
6.7 Market value . . . 29
6.8 How to sell it . . . 29
6.9 Risks & mitigations . . . 29
6.10 90-day roadmap . . . 29

7 Guide 4 — AI Academic Writing & Publication Agent 30
7.1 "From Data to Manuscript" . . . 30
7.2 The problem . . . 30
7.3 The solution . . . 30
7.4 Architecture . . . 31
7.5 Build steps . . . 31
7.6 Product extensions . . . 32
7.7 Market value . . . 32
7.8 How to sell it . . . 32
7.9 Risks & mitigations . . . 32
7.10 90-day roadmap . . . 32

8 Guide 5 — AI PhD Supervisor Agent 34
8.1 "Virtual Research Mentor" . . . 34
8.2 The problem . . . 34
8.3 The solution . . . 34
8.4 Architecture — memory is the product . . . 35
8.5 Build steps . . . 35
8.6 Product extensions . . . 35
8.7 Market value . . . 35
8.8 How to sell it . . . 36
8.9 Risks & mitigations . . . 36
8.10 90-day roadmap . . . 36

9 Guide 6 — The AI Research OS (Platform Vision) 37
9.1 One platform, five agents, the whole research lifecycle . . . 37
9.2 What it is . . . 37
9.3 Why a platform beats five point tools . . . 37
9.4 Platform architecture (Phase 2–3) . . . 38
9.5 Institutional requirements (what universities will ask for) . . . 38
9.6 Pricing the platform . . . 38
9.7 Go-to-market for the platform . . . 38
9.8 Moats . . . 39
9.9 Team to build it (first 12 months) . . . 39

10 Guide 7 — Go-to-Market, Pricing & Selling Playbook 40
10.1 How to actually make money from research agents . . . 40
10.2 The three customers (and who pays) . . . 40
10.3 Pricing (starting point; adjust with data) . . . 40
10.4 Channels . . . 41
10.5 The pilot playbook (institutions) . . . 41
10.6 Sales collateral you need . . . 41

10.7 Objections & answers . . . 41
10.8 12-month revenue model (illustrative) . . . 42
10.9 Legal & compliance checklist . . . 42

III Part III — Learning to build AI agents 43

11 What to Learn to Build AI Agents (Roadmap) 44
11.1 Stage 0 — Python you actually need (2–3 weeks) . . . 44
11.2 Stage 1 — LLM fundamentals (1–2 weeks) . . . 44
11.3 Stage 2 — Tools & agents (2–3 weeks) . . . 45
11.4 Stage 3 — Research-domain tools (1–2 weeks) . . . 45
11.5 Stage 4 — Productising (3–4 weeks) . . . 45
11.6 Stage 5 — Advanced (ongoing) . . . 45
11.7 Suggested weekly plan (8 weeks) . . . 45

12 Glossary — Major Terms in AI Agent Building 47
12.1 Core LLM terms . . . 47
12.2 Agent terms . . . 48
12.3 Research-domain terms . . . 48

13 How These AI Agents Work (Architecture & Patterns) 50
13.1 The one-sentence model . . . 50
13.2 Two ways to run a loop . . . 50
13.3 The five-step template every agent follows . . . 51
13.4 Structured outputs — the backbone . . . 51
13.5 Grounding & anti-hallucination . . . 51
13.6 Memory . . . 51
13.7 Multi-agent patterns you will see . . . 52
13.8 Cost & quality levers . . . 52
13.9 Evaluation — how you know it works . . . 52
13.10 From CLI to product . . . 52

14 Resources & Community 53
14.1 Learn with Codanics (start here) . . . 53
14.2 APIs used in the code (all free tiers) . . . 53
14.3 Python libraries used . . . 53
14.4 Methodology references . . . 53
14.5 Comparable products (study their pricing & UX) . . . 53
14.6 Tools for the next steps . . . 54

IV Part IV — The code 55

15 The code — setup and shared pattern 56
15.1 `02_agents_code/README.md` . . . 56
15.2 requirements.txt . . . 57
15.3 .env.example . . . 58
15.4 setup_venv.ps1 . . . 58
15.5 setup_venv.sh . . . 59
15.6 .gitignore . . . 59

16 Code: Agent 1 — Research Discovery 60
16.1 README . . . 60
16.2 Source files . . . 61

17 Code: Agent 2 — Literature Review 72
17.1 README . . . 72
17.2 Source files . . . 73

18 Code: Agent 3 — Research Experiment **86**
18.1 README 86
18.2 Source files 87

19 Code: Agent 4 — Academic Writing & Publication **98**
19.1 README 98
19.2 Source files 99

20 Code: Agent 5 — PhD Supervisor **108**
20.1 README 108
20.2 Source files 109

1 Five AI Research Agents



Build the AI Research OS.

For **research and academic writing**, the biggest opportunity is not another ChatGPT wrapper. It is **AI research agents that automate the complete research lifecycle** — idea discovery → literature review → experimentation → writing → publication.

This book is the complete product-development kit for five such agents: the **guides**, the **runnable Python code**, the **market and pricing analysis**, and the **learning path** to build them yourself.

Read it your way — this book is published in three formats: **HTML** (this site), **PDF** and **EPUB**. You can also use the download icon in the top bar at any time.

1.1 The five agents

1AI Research Discovery Agent *"Your AI Research Scientist"* — mines arXiv, PubMed, Semantic Scholar and Crossref, finds research gaps, generates ranked PhD topics and writes the proposal.

2AI Literature Review Agent *"Autonomous Systematic Review Scientist"* — PRISMA search → screening → evidence extraction → a full literature review draft, with an audit trail.

3AI Research Experiment Agent *"Virtual Research Assistant"* — experimental design, QC, statistics, machine learning and a reproducible analysis notebook.

4AI Academic Writing & Publication Agent *"From Data to Manuscript"* — manuscript drafting, journal matching, simulated peer review and a point-by-point response letter.

5AI PhD Supervisor Agent *"Virtual Research Mentor"* — a 3-year roadmap, weekly progress evaluation, critical review and defence preparation, backed by long-running memory.

1.2 Who this book is for

- **PhD scholars and researchers** who want to cut months off literature reviews and manuscripts.
- **Python learners** who want to build real, sellable AI agents rather than toy chatbots.
- **Founders and product people** looking for a validated, underserved market.
- **Universities, research labs and R&D departments** evaluating an institutional AI research platform.

1.3 How the book is organised

- **Part I** — a five-member expert council debates what to build, for whom and how to sell it; then the market overview.
- **Part II** — one guide per agent: problem, solution, architecture, build steps, market value, how to sell, risks and a 90-day roadmap; plus the platform vision and the go-to-market playbook.
- **Part III** — what to learn to build AI agents, a glossary of the major terms, and how these agents actually work.
- **Part IV** — the complete, commented Python source of all five agents.

1.4 How to run the code

Each agent is a **self-contained folder** — copy one out and it still runs. All five share one virtual environment.

```
cd 02_agents_code
.\setup_venv.ps1                # creates .venv and installs the requirements
notepad .env                     # add your ANTHROPIC_API_KEY
.\.venv\Scripts\Activate.ps1
python agent_01_research_discovery\main.py --topic "AI-based drought prediction in wheat"
```

Agents 1 and 2 accept `--offline` (built-in sample papers) and Agent 3 accepts `--no-llm` (the numeric pipeline only), so you can try the book without an API key.

1.5 The design philosophy

> **Agents draft; humans decide.** Every citation an agent produces comes from a retrieved record, every number comes from code rather than from the model, and every output carries an AI-assistance notice. That principle runs through all five agents in this book.

1.6 Codanics links

🎓 Build AI Agents course 🆓 Free course playlist ▶️ YouTube @codanics 💬 WhatsApp channel 🌐 codanics.com

Codanics — Python ka Chilla: Build AI Agents Dr. Muhammad Aammar Tufail · Founder/CEO, Codanics Learn AI, data science and bioinformatics — in Urdu/Hindi and English.

Course Website YouTube WhatsApp Free course

Part I

Part I — The Council & the Opportunity

2 The Council of Five — Deliberation & Decision

Purpose of this document. Before writing a single line of code, we convened a council of five expert personas to decide *what* to build, *in which order*, *for whom*, and *how to sell it*. Everything else in this repository (guides, code, learning material, book) follows the decisions taken here.

2.1 The council members

Seat	Persona	Background	What they guard
1	**Dr. Ayesha — the Research Scientist**	12 years in plant genomics & bioinformatics, supervises 6 PhD students, publishes in Q1 journals	Scientific rigour, real researcher pain points, reproducibility, ethics of AI in research
2	**Bilal — the AI Agent Architect**	Built production LLM systems (RAG, multi-agent orchestration, tool use, evals)	Architecture, tool design, reliability, cost, latency, evaluation
3	**Sara — the Product & Go-to-Market Lead**	Ex-EdTech/SaaS founder, sold to universities in Pakistan, GCC and UK	Pricing, positioning, procurement cycles, who signs the cheque
4	**Prof. Khan — the University Administrator**	Dean of Research at a public-sector university, sits on HEC/ORIC committees	Institutional buying, compliance, data governance, training needs
5	**Hamza — the Python Educator (Codanics voice)**	Teaches "Python ka Chilla: Build AI Agents"; thousands of students	Teachability, clean code, step-by-step learning path, community building

---

2.2 The question put to the council

"We have five candidate AI research agents (Discovery, Literature Review, Experiment, Writing & Publication, PhD Supervisor) and a vision of an *AI Research OS*. What do we build first, how do we build it, how do we sell it, and how do we teach it?"

2.3 Round 1 — Each member's opening position

2.3.1 Dr. Ayesha (Research Scientist)

- The **Literature Review Agent** solves the most painful, most measurable problem: a systematic review is 6–12 months of labour. If we cut it to 2–4 weeks *with an auditable PRISMA trail*, every graduate school will want it.

- Warning: researchers will not trust a black box. Every claim must be **traceable to a real paper (DOI/arXiv ID)**. Hallucinated citations = instant loss of trust.
- The **Experiment Agent** is high value but domain-specific (RNA-seq ≠ soil science ≠ NLP). Build it as a *framework* with pluggable pipelines, not a one-size-fits-all.

2.3.2 Bilal (AI Agent Architect)

- All five agents share ~70% of infrastructure: an LLM client, paper-search tools (arXiv, PubMed, Semantic Scholar, Crossref), structured-output extraction, a planner → worker → critic loop, and a report writer.
- Recommendation: **build each agent as an independent, copy-able folder** (students must be able to run one agent without the others), but follow the *same internal pattern* so learning transfers.
- Use **structured outputs (Pydantic)** everywhere we need tables (screening decisions, extraction tables, reviewer comments). Free-text for prose sections only.
- Insist on **evals** from day one: a small "golden set" of topics with expected papers/gaps.

2.3.3 Sara (Product & GTM)

- Individual researchers pay $20–50/month; institutions pay $20k–$500k/year. **Sell to institutions, market to individuals.** Individuals become the champions who push procurement.
- Sequence: *Literature Review Agent* (fastest "wow", clear ROI) → *Discovery Agent* (top-of-funnel, viral demo) → *Writing & Publication Agent* (recurring revenue: every paper) → *PhD Supervisor* (retention) → *Experiment Agent* (enterprise/pharma upsell).
- Emerging markets (Pakistan, GCC, South/South-East Asia, Africa) are underserved. Pricing must be **tiered by region**; HEC-affiliated universities in Pakistan can be reached via ORICs (Offices of Research, Innovation & Commercialization).

2.3.4 Prof. Khan (University Administrator)

- Universities buy on: (1) plagiarism/ethics safety, (2) data privacy (student data stays in-country or on-prem), (3) training & support, (4) a report we can show to accreditation bodies.
- A **PhD Supervisor Agent** would be politically sensitive if it *replaces* supervisors — position it as a **supervisor's assistant** and a **student's study coach**.
- Procurement cycle is 3–9 months. Offer a **free pilot for one department** with a success metric (e.g., "reduce time-to-literature-review by 60%").

2.3.5 Hamza (Educator, Codanics)

- Students must be able to run every agent with **one API key, one `.venv`, and one `python main.py` command**.
- Each agent should teach one new concept: tools & search APIs (Discovery), structured outputs & pipelines (Lit Review), code + data + stats (Experiment), multi-persona simulation (Writing), memory & long-running state (Supervisor).
- All materials must be **branded and free to learn** (YouTube playlist + WhatsApp channel), with the paid course as the deep-dive. Everything ends up in a **Quarto book** so it doubles as course material.

2.4 Round 2 — Debate & resolutions

Tension	Resolution
"Build the whole Research OS now" vs. "ship one agent"	**Ship the five agents as independent modules first**, with a thin orchestrator (`AI Research OS`) described in the guides as Phase 2. Independent modules are easier to sell, teach and test.
Free-text LLM output vs. structured data	**Structured outputs (Pydantic) for every decision/table**; free text only for narrative sections. This is non-negotiable for trust and for building a UI later.
Which LLM / provider	Default to the **Anthropic Claude API** (`claude-opus-5`, adaptive thinking, streaming). Keep a single `llm.py` per agent so swapping providers is a one-file change.
Real search APIs vs. mock data	**Real, free APIs** (arXiv, Semantic Scholar, PubMed E-utilities, Crossref) with graceful offline fallbacks so students without internet access can still run demos.
Hallucinated citations	Every citation must originate from a **tool result**, never from the model's memory. The writer agents receive *only* the retrieved paper list and are told to cite only from it.
Supervisor agent ethics	Reframe as **"Virtual Research Mentor" — assistant to supervisors, coach to students.** Include an explicit "AI limitations & academic integrity" notice in outputs.
Experiment agent scope	Ship a **general tabular/omics-style pipeline** (QC → differential analysis → ML → stats → methods text) using synthetic demo data, designed to be extended with domain plugins.

2.5 The decision

2.5.1 What we build (in this repository)

1. **Five independent Python agents** in `02_agents_code/`, each with its own `README.md`, `main.py`, `agents.py`, `tools.py`, `llm.py` and `outputs/`.
2. **One guide per agent** in `01_guides/` covering: problem, solution, architecture, workflow, build steps, market value, pricing, how to sell, risks, roadmap.
3. **A learning pack** in `03_learning/`: what to learn, glossary of major terms, how the agents work (patterns), resources.
4. **A Quarto book** in `04_quarto_book/` that stitches everything into a publishable book.

2.5.2 Build order (product roadmap)

Phase	Months	Deliverable	Success metric
0	0–1	This repo: 5 CLI agents + guides + book	Runs end-to-end on one API key
1	1–3	Literature Review Agent as a web app (Streamlit/FastAPI) + PDF ingestion	10 pilot users, 60% time saved
2	3–6	Discovery + Writing agents in the same app; user accounts; export to Word/LaTeX	100 paying individuals

Phase	Months	Deliverable	Success metric
3	6–12	Institutional edition: SSO, on-prem/VPC option, admin dashboard, usage reports	2 university licences
4	12–18	"AI Research OS" orchestrator: idea → review → experiment → paper → submission	$250k ARR

2.5.3 Who we sell to first

- **Beachhead:** PhD scholars & early-career faculty in agriculture, biology, medicine and computer science in Pakistan, GCC and South Asia (large, price-sensitive, underserved, fast to adopt).
- **Cheque-writers:** ORIC directors, Deans of Research, research-lab PIs, R&D heads in pharma/agri companies.

2.5.4 How we make money

Tier	Price	Includes
Free / Student	$0	3 reviews/month, watermark, community support
Individual Pro	$20–30 / month (regional pricing)	Unlimited runs, exports, priority models
Lab	$2k–5k / year	10 seats, shared library, admin
Institution	$20k–100k / year	Unlimited seats, SSO, training, support, reports
Enterprise (pharma / agri R&D)	$100k–500k / year	On-prem/VPC, custom pipelines, SLAs

2.5.5 Guiding principles (signed by all five)

1. **Traceability over eloquence** — no citation without a source ID.
2. **Independent modules, shared patterns** — copy one folder, it still runs.
3. **Structured first** — Pydantic models for every decision.
4. **Teach while building** — every file is readable by a student of "Python ka Chilla".
5. **Human in the loop** — agents draft, humans decide; especially for supervision and publication.

2.6 Council sign-off

Member	Vote	Note
Dr. Ayesha	✅	"Ship the Literature Review Agent first, and make PRISMA the backbone."
Bilal	✅	"Same pattern in all five: Plan → Retrieve → Structure → Critique → Write."
Sara	✅	"Institutions pay; individuals evangelise. Pilot with one ORIC."

Member	Vote	Note
Prof. Khan	✅	"Data privacy and academic-integrity notices in every output."
Hamza	✅	"One `.venv`, one key, one command per agent. Everything goes in the book."

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

3 Guide 0 — The Opportunity: AI Research Agents & the "AI Research OS"

3.1 Why research is the best vertical for AI agents

Research is **text-heavy, process-heavy and repetitive** — exactly what LLM agents are good at:

Research task	Today (manual)	With an agent
Finding papers & gaps	Weeks–months of reading	Hours, with citation graph & gap statement
Systematic / literature review	6–12 months	2–4 weeks with PRISMA trail
Experiment design & analysis	Needs statistician + bioinformatician	Guided pipeline + auto-generated notebook
Writing & submitting a paper	2–6 months, many rejections	Draft in days; journal matching; simulated review
PhD supervision	1 meeting per month if lucky	Weekly structured feedback

The million-dollar opportunity is **not another ChatGPT wrapper**. It is an **agent that owns a complete workflow**, keeps state, uses real research tools (arXiv, PubMed, Semantic Scholar, Crossref, statistical libraries), and produces auditable outputs (tables, PRISMA diagrams, notebooks, manuscripts).

3.2 Market size (order-of-magnitude, for planning)

Segment	Global count (approx.)	Willingness to pay
Universities & HEIs	~25,000	$20k–500k / year for research tools (they already buy Turnitin, EndNote, Scopus, SciVal)
PhD students	~3–4 million enrolled	$10–50 / month
Active researchers / faculty	~9 million	$20–50 / month; labs $2k–5k / year
Corporate R&D (pharma, agri, engineering)	tens of thousands of labs	$100k–1M / year
Emerging-market universities (Pakistan, GCC, South Asia, Africa)	~3,000+	Underserved; regional pricing; strong policy push for research output

Comparable products & prices (to anchor pricing): Elicit, Consensus, Scite, SciSpace, Research Rabbit, Litmaps, Paperpal, Writefull — individual plans typically $10–50/month; institutional deals $10k–200k/year.

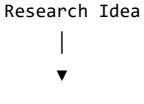
3.3 The five agents at a glance

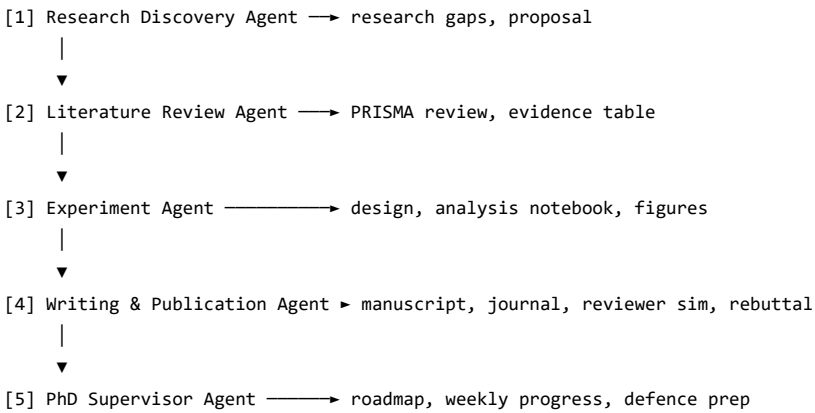
#	Agent	Core loop	Teaches you	Guide
1	Research Discovery	Search → cluster → gap → idea → proposal	Tool use, search APIs, citation networks	`01_research_discovery_agent.md`
2	Literature Review	PRISMA search → screen → extract → write	Structured outputs, pipelines, auditability	`02_literature_review_agent.md`
3	Research Experiment	Plan → analyse → model → test → methods	Code + data + statistics agents	`03_research_experiment_agent.md`
4	Writing & Publication	Draft → match journal → simulate review → rebut	Multi-persona simulation, critique loops	`04_academic_writing_publication_agent.md`
5	PhD Supervisor	Roadmap → weekly check-in → critique → slides	Memory, long-running state, coaching	`05_phd_supervisor_agent.md`

3.4 Ranking (from the council)

Rank	Agent	Market	Revenue potential	Why
🥇	Literature Review Agent	Massive	$$$$$	Everyone needs it; measurable ROI; easiest to demo
🥈	Research Discovery Agent	Massive	$$$$$	Top-of-funnel; viral; feeds all other agents
🥉	Writing & Publication Agent	Very high	$$$$	Recurring — every paper; upsell to institutions
4	Experiment Agent	High	$$$$$	Highest enterprise value (pharma/agri) but domain-specific
5	PhD Supervisor Agent	High	$$$$	Retention & institutional stickiness

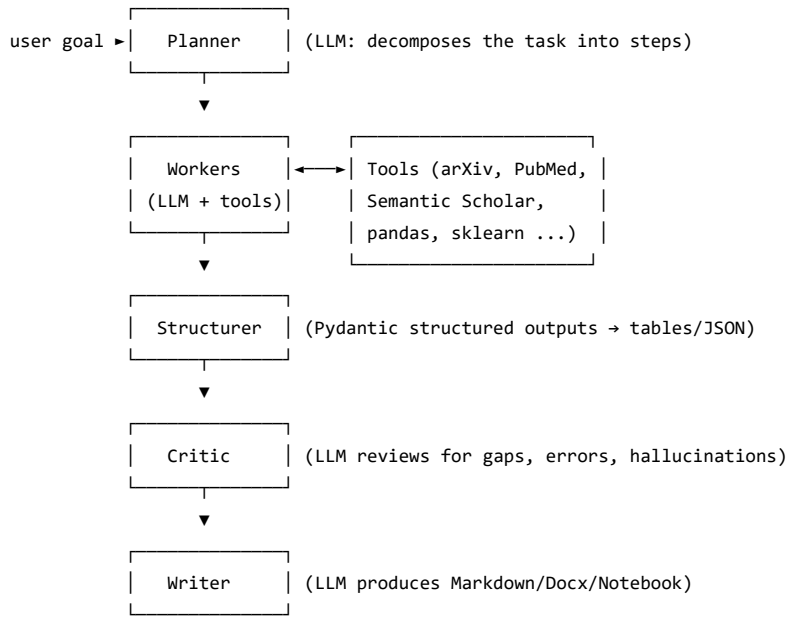
3.5 The AI Research OS (Phase 2 vision)





Think **"Cursor + ChatGPT + EndNote + a research assistant + a bioinformatics analyst — for scientists."** A university could pay **$50k–500k / year** for a platform that owns this whole pipeline.

3.6 Common architecture (all five agents share it)



3.7 Where the money is (summary)

- **Individuals:** $20–30/month × thousands of PhD students = steady base revenue and word-of-mouth.
- **Institutions:** $20k–100k/year × even 10 universities = $200k–1M ARR.
- **Enterprise R&D:** $100k–500k/year per pharma/agri company for private deployments and custom pipelines.
- **Services:** training workshops, custom pipeline development, data-migration — often 20–30% of early revenue.

See `07_go_to_market_and_pricing.md` for the full sales playbook.

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

Part II

Part II — Guides: one per agent

4 Guide 1 — AI Research Discovery Agent

4.1 "Your AI Research Scientist"

4.2 The problem

Researchers spend **months** finding:

- novel research gaps,
- the right papers among thousands,
- emerging topics and methods,
- the key authors and groups in a field.

Most of them still read hundreds of papers manually, keep notes in Word files, and *hope* their idea is novel.

4.3 The solution

An autonomous agent that, given a one-line research interest, continuously monitors the literature (arXiv, PubMed, Semantic Scholar, Crossref — extendable to IEEE Xplore, Springer, Elsevier, Google Scholar) and builds a **personal research-intelligence system**:

1. **Literature Mining Agent** — finds the latest N papers, builds a citation network, finds key authors and trends.
2. **Research Gap Agent** — reads the abstracts and writes explicit gap statements ("Existing studies predict drought stress using RGB imaging, but few integrate hyperspectral + genomic data").
3. **Idea Generator Agent** — proposes ranked PhD/grant topics with novelty and feasibility scores.
4. **Proposal Writer Agent** — turns the best idea into a research proposal: title, novelty statement, research questions, hypotheses, objectives, methodology sketch, timeline.

4.4 Example

Input

> "I want to work on AI-based drought prediction in wheat."

Output (Markdown report + JSON)

- 50–500 papers with IDs, years, venues, citation counts
- Trend summary (topics per year, rising methods)
- 5–8 research gaps with supporting paper IDs
- 5 ranked topic ideas, e.g. "Multimodal AI for drought-resilience prediction", "Explainable ML for genomic selection"
- A 2–3 page research proposal for the top idea

4.5 Architecture

```
topic ──► [Search tools] arXiv / Semantic Scholar / PubMed / Crossref
              │  (real APIs, free)
              ▼
        Literature Mining Agent   → PaperSet (Pydantic), trends, key authors
              │
              ▼
        Research Gap Agent        → GapAnalysis (list of gaps + evidence IDs)
              │
              ▼
        Idea Generator Agent      → IdeaList (title, novelty, feasibility, impact)
              │
              ▼
        Proposal Writer Agent     → proposal.md
```

Key design rules:

- **Citations only from tool results.** The model receives the retrieved papers and is instructed to cite only their IDs.
- **Structured outputs** (Pydantic) for papers, gaps and ideas so they can be rendered as tables or stored in a DB.
- **Idempotent runs**: every run writes `outputs/<slug>/papers.json`, `gaps.json`, `ideas.json`, `proposal.md`.

4.6 Build steps (what the code does)

Step	File	What you learn
1	`tools.py`	Calling arXiv, Semantic Scholar, PubMed & Crossref; normalising results into one `Paper` model
2	`llm.py`	A thin Claude client with streaming + structured outputs
3	`agents.py`	Four single-responsibility agents with clear system prompts
4	`main.py`	A CLI orchestrator (`--topic`, `--max-papers`, `--years`) that runs the pipeline and saves outputs

Run:

```
python agent_01_research_discovery\main.py --topic "AI-based drought prediction in wheat" --max-papers 60
```

4.7 Extending it (product features)

- **Continuous monitoring**: schedule weekly runs (cron / Managed Agents scheduled deployments) and diff the new papers → "What's new this week in your field" email.
- **Citation network graph**: use Semantic Scholar's references/citations endpoints + `networkx` → find the "bridge" papers.
- **Personal library**: store papers in SQLite/Postgres + vector index for RAG-style Q&A.
- **Web UI**: Streamlit for the MVP, then FastAPI + React.
- **More sources**: Scopus/IEEE/Springer via institutional API keys (sold as an institutional feature).

4.8 Market value

Customer	Pain	Willing to pay
PhD students	"Is my topic novel?"	$10–30/month
Supervisors / PIs	Need topics for new students, grant ideas	$30–50/month or lab plan
Universities / ORICs	Improve proposal quality & research output	$20k–100k/year
R&D departments	Technology scouting, competitor tracking	$50k–200k/year

Estimated TAM for discovery tools alone: **$1–2B** (research-intelligence + scholarly search market).

4.9 How to sell it

1. **Free "Is my topic novel?" check** on the website — users paste a topic, get 3 gaps and 3 ideas (watermarked). Capture email.
2. **Content marketing** — YouTube walkthroughs ("I found a PhD topic in 10 minutes"), WhatsApp community, university seminars.
3. **Supervisor referrals** — supervisors get a free seat for every 5 students.
4. **ORIC pilots** — offer a department-wide 3-month pilot; report "proposals generated / accepted".
5. **Upsell** to the Literature Review Agent (natural next step) and the full Research OS.

4.10 Risks & mitigations

Risk	Mitigation
Hallucinated papers	Only tool-retrieved papers may be cited; verify DOIs via Crossref
Shallow gap analysis	Feed abstracts (not just titles); allow full-text PDFs in Pro tier
API rate limits	Cache results; polite delays; institutional API keys
"Ideas are generic"	Ask for *feasibility, required data, required equipment* per idea; score novelty against retrieved set

4.11 90-day roadmap

- Weeks 1–2: CLI (this repo) → Weeks 3–6: Streamlit UI + PDF upload → Weeks 7–10: weekly monitoring & email digests → Weeks 11–13: citation graph & lab plan.

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

5 Guide 2 — AI Literature Review Agent

5.1 "Autonomous Systematic Review Scientist"

5.2 The problem

A systematic review takes **6–12 months**. Researchers manually:

- search 3–6 databases with hand-crafted queries,
- screen hundreds–thousands of titles/abstracts,
- read full texts and extract data into spreadsheets,
- build PRISMA flow diagrams and evidence tables,
- write the review, the gap analysis and the discussion.

Every graduate student does at least one; every clinical guideline, meta-analysis and grant proposal needs one.

5.3 The solution

A four-stage agent pipeline that follows the **PRISMA** methodology and keeps an **auditable trail** of every decision:

Stage	Agent	Output
1	**Search Agent**	Search strategy (queries per database, inclusion/exclusion criteria) + retrieved records
2	**Screening Agent**	For each record: `include / exclude / maybe` + a one-line reason ("Excluded: dataset unavailable")
3	**Knowledge Extraction Agent**	Evidence table: Paper · Dataset · Method · Sample size · Result · Limitations
4	**Writing Agent**	Literature review draft: introduction, methods (PRISMA), results (themes), gaps, discussion, references

5.3.1 Example

Input: "Review AI methods for plant disease detection"

Output: a review draft (Markdown → Word/PDF) with a PRISMA flow summary, an evidence table of 40–200 papers, thematic synthesis and research gaps.

5.4 Architecture

```
question → Search Agent → queries → [arXiv | PubMed | Semantic Scholar | Crossref]
                                          |
                                          ▼   records (deduplicated by DOI/title)
                                    Screening Agent (structured: decision + reason)
                                          |  include
                                          ▼
                                    Extraction Agent (structured: EvidenceRow)
                                          |
                                          ▼
                                    Writing Agent  → review.md + prisma.json + evidence.csv
```

Non-negotiables

- **PRISMA counts** (identified → deduplicated → screened → excluded → included) are computed by *code*, not by the LLM.
- **Every included paper** appears in the evidence table with its ID; the writer cites only those IDs.
- **Screening is batched** (e.g. 10 abstracts per call) with structured outputs → cheap and reproducible.
- **Human override**: the CSV can be edited and the writer re-run (`--from-evidence`).

5.5 Build steps

Step	File	What you learn
1	`tools.py`	Multi-database search + deduplication
2	`prisma.py`	Counting and rendering the PRISMA flow (pure Python)
3	`agents.py`	Search-strategy, batch screening, extraction, writing agents (all structured except the writer)
4	`main.py`	CLI: `--question`, `--max-records`, `--from-evidence`

Run:

```
python agent_02_literature_review\main.py --question "AI methods for plant disease detection" --max-records 80
```

5.6 Product extensions

- **Full-text PDF ingestion** (pypdf / GROBID) for extraction from methods/results sections.
- **Dual screening** with two LLM "reviewers" + conflict resolution → Cohen's kappa report.
- **Meta-analysis module**: pooled effect sizes with `statsmodels`/`metafor`-like forest plots.
- **Reference manager export**: BibTeX/RIS, Zotero sync.
- **Templates**: Cochrane, PRISMA-ScR (scoping), JBI, engineering (SLR by Kitchenham).
- **Team workspace**: shared reviews, assignments, comments.

5.7 Market value

Customer	Value delivered	Price point
PhD / MS students	Thesis chapter 2 in weeks not months	$20–30/month

Customer	Value delivered	Price point
Clinical & public-health researchers	Guideline-grade reviews, faster	$50/month; lab $3–5k/year
Universities (graduate schools, libraries)	Faster completions, more publications	$30k–100k/year
Pharma / CROs / HTA bodies	Evidence synthesis at scale	$100k–500k/year

Comparable tools: Covidence, Rayyan, DistillerSR, Elicit — Covidence alone is used by 100k+ reviewers and priced from ~$250/review to institutional licences in the tens of thousands.

5.8 How to sell it

1. **"Chapter 2 in a week" campaign** for PhD students (YouTube demo + WhatsApp channel).
2. **Library partnership**: university libraries already licence Covidence/Rayyan — position as the AI-native alternative at lower cost.
3. **Journal-club workshops**: run a free 2-hour "systematic review with AI" workshop at a department; convert to a lab plan.
4. **Evidence-synthesis services** for pharma/NGOs (services revenue while the product matures).
5. **Accreditation angle**: universities need publication counts — sell the *output metric*.

5.9 Risks & mitigations

Risk	Mitigation
Missed papers (recall)	Multiple databases, query expansion by the Search Agent, snowballing via citations
Wrong screening decisions	Dual-LLM screening, human review of `maybe`, kappa report
Extraction errors	Show evidence quote for every extracted field; allow edit-and-rerun
Journals/ethics policies on AI use	Transparent "AI-assisted" statement generator; human-in-the-loop by design

5.10 90-day roadmap

- Weeks 1–2: CLI (this repo) → Weeks 3–6: Streamlit app + PDF upload + export → Weeks 7–10: dual screening & kappa → Weeks 11–13: team workspace & Zotero.

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

6 Guide 3 — AI Research Experiment Agent

6.1 "Virtual Research Assistant"

6.2 The problem

Researchers — especially in biology, medicine and agriculture — struggle with:

- **experimental design** (controls, replicates, power),
- **data analysis** (QC, normalisation, differential analysis),
- **coding** (Python/R pipelines),
- **statistics** (which test, multiple-testing correction, effect sizes),
- **reporting** (figures, tables, a defensible Methods section).

A lab typically waits weeks for a bioinformatician or statistician.

6.3 The solution

An agent that manages the experiment from plan to report:

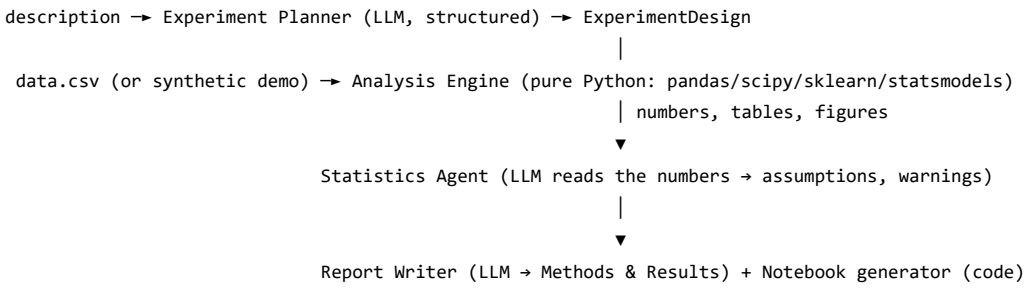
Agent	Role	Output
Experiment Planner	Turns a one-line description into a design: groups, replicates, covariates, power notes, analysis plan	`design.json`
Analysis (Bioinformatics/Data) Agent	Runs QC, normalisation, differential analysis (e.g. control vs. treatment) using pandas/scipy	tables + figures
ML Agent	Trains baseline models (logistic regression, random forest), cross-validation, feature importance	`ml_results.json`, figures
Statistics Agent	Checks assumptions, corrects for multiple testing, flags bias/leakage, interprets effect sizes	`stats_report.md`
Report Writer	Produces the Methods + Results text and a reproducible notebook	`methods_results.md`, `analysis_notebook.ipynb`

6.3.1 Example

Input: "I have RNA-seq data from heat-stressed rice (control vs heat, n=3 each)."

Output: design (control vs heat stress, n=3), QC summary, differential-expression table (log2FC, p, FDR), volcano plot, a random-forest classifier with CV accuracy, a statistics sanity report, and a Methods/Results section — plus a Jupyter notebook that reproduces everything.

6.4 Architecture — *code does the math, the LLM does the reasoning*



Why this split matters: LLMs are unreliable at arithmetic and p-values. All statistics are computed by deterministic Python; the LLM only *plans*, *interprets* and *writes*. This keeps results reproducible and defensible.

6.5 Build steps

Step	File	What you learn
1	`data.py`	Loading real CSVs or generating a synthetic omics-style dataset (genes × samples)
2	`analysis.py`	QC, normalisation (log/CPM-like), t-tests per feature, BH-FDR, volcano plot, RF classifier with CV
3	`agents.py`	Planner (structured), Statistics reviewer (structured), Report writer
4	`notebook.py`	Generating a runnable `.ipynb` with `nbformat`
5	`main.py`	CLI: `--description`, `--data path.csv`, `--group-col`, `--demo`

Run (demo data):

```
python agent_03_research_experiment\main.py --description "RNA-seq of heat-stressed rice, control vs heat, n=3" --demo
```

Run (your data):

```
python agent_03_research_experiment\main.py --description "..." --data mydata.csv --group-col condition
```

6.6 Product extensions

- **Domain plugins**: RNA-seq (DESeq2 via rpy2 / pydeseq2), GWAS, metabolomics, field trials (ANOVA/mixed models), clinical trials (survival analysis), social science (regressions/SEM).
- **Sandboxed code execution** so the agent can write and run custom analysis code safely (Docker / Claude code-execution tool).

- **Figure styling** to journal specs; automatic figure legends.
- **Lab data connectors**: LIMS, Google Sheets, S3, sequencing cores.
- **Power analysis & sample-size calculator** at design time.

6.7 Market value

Customer	Value	Price
Biology / medical / agriculture labs	Replace weeks of waiting for an analyst	$50–100/month per researcher; lab $5k/year
Core facilities & bioinformatics units	Handle 5× the projects	$20k–50k/year
Universities	Standardised, reproducible analysis for theses	$50k–150k/year
Pharma / agri-biotech R&D	Private deployment, custom pipelines, audit trails	$150k–500k/year

This is the **highest-value enterprise agent** in the set because the alternative is hiring specialists.

6.8 How to sell it

1. **Free "analyse my CSV" demo** (synthetic or uploaded, capped size).
2. **Core-facility partnership**: give the facility a dashboard; they resell analysis to labs.
3. **Grant-budget line**: position at $5k/year as a "bioinformatics support" line item in grants.
4. **Pharma/agri pilots**: 3-month pilot on one dataset with a reproducibility report.
5. **Training**: "Data analysis for biologists with AI agents" workshops (Codanics-style) → lab plans.

6.9 Risks & mitigations

Risk	Mitigation
Wrong statistical test chosen	Statistics Agent checks assumptions and explains alternatives; rule-based defaults per data type
Data leakage in ML	Explicit CV pipeline; agent flags leakage patterns
Users trust numbers blindly	Every result carries method, n, test, correction; report includes "limitations"
Data privacy (clinical)	On-prem / VPC option; no raw data sent to the LLM — only summary statistics

6.10 90-day roadmap

- Weeks 1–2: CLI with synthetic pipeline (this repo) → Weeks 3–6: CSV/Excel upload UI, figure export → Weeks 7–10: RNA-seq & field-trial plugins → Weeks 11–13: sandboxed custom code + notebook editor.

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

7 Guide 4 — AI Academic Writing & Publication Agent

7.1 "From Data to Manuscript"

7.2 The problem

Researchers have results but struggle with:

- writing a coherent paper (especially non-native English writers),
- formatting for a specific journal,
- choosing the right journal (scope, impact, cost, acceptance odds),
- surviving peer review and writing point-by-point responses.

A paper often takes 3–6 months from results to submission, and 40–60% of first submissions are rejected.

7.3 The solution

A **multi-agent publishing assistant**:

Agent	Role	Output
Manuscript Agent	Builds the IMRaD manuscript from the researcher's notes/results: title, abstract, introduction, methods, results, discussion, conclusion	`manuscript.md`
Journal Matching Agent	Scores candidate journals on scope fit, impact, open-access cost, turnaround, acceptance probability	`journal_matches.json`
Reviewer Simulation Agent	Two or three simulated reviewers with different personas (methodologist, domain expert, statistician) produce numbered comments and a recommendation	`reviews.json`
Revision Agent	Produces a point-by-point response letter and a revised manuscript addressing each comment	`response_letter.md`, `manuscript_v2.md`

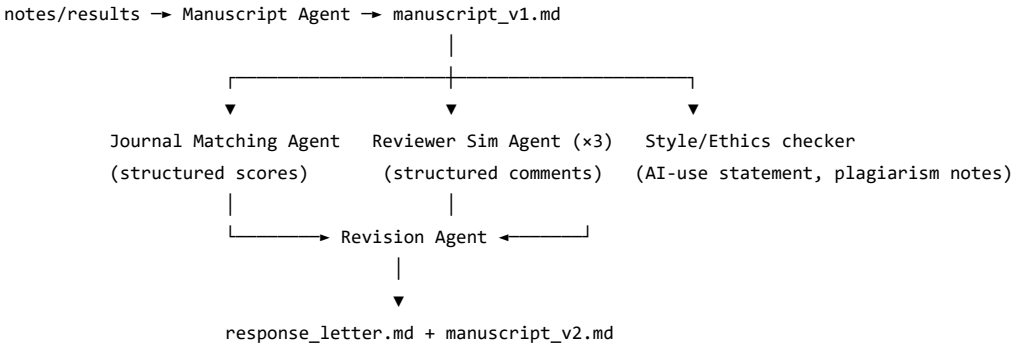
7.3.1 Example

Input: a `notes.md` file with the study summary, key results and a few numbers.

Output:

- A full manuscript draft with an "AI-assisted writing" statement.
- Top-5 journal recommendations with reasons.
- Reviewer 1: "Your methodology lacks validation." Reviewer 2: "Explain dataset selection." (with severity).
- A response letter: "We thank the reviewer for this comment. We have added …" and a revised draft.

7.4 Architecture



Design rules

- The Manuscript Agent may **not invent results or citations** — it writes only from the provided notes and (optionally) a references list from Agent 1/2.
- Reviewer personas are **parameterised** (harsh/moderate; methods/domain/stats) → diverse critique.
- The Revision Agent must **quote each comment** and mark where the change was made.

7.5 Build steps

Step	File	What you learn
1	`journals.py`	A small journal knowledge base (name, scope, impact band, OA fee, typical review time) — replace with Scimago/DOAJ/OpenAlex data in production
2	`agents.py`	Manuscript writer, journal matcher (structured), reviewer simulator (structured), revision agent
3	`main.py`	CLI: `--notes notes.md`, `--field`, `--reviewers 3`

Run:

```
python agent_04_academic_writing\main.py --notes agent_04_academic_writing\sample_notes.md --field "plant science"
```

7.6 Product extensions

- **Live journal data** from OpenAlex/Scimago/DOAJ APIs; predatory-journal blacklist.
- **Formatting engines**: Word/LaTeX templates per journal (Elsevier, Springer, IEEE, MDPI, Frontiers).
- **Reference management**: BibTeX/Zotero, citation-style rendering (APA, Vancouver).
- **Cover letter, highlights, graphical abstract prompts**.
- **Grammar & clarity pass** tuned for academic English (a key value for non-native writers).
- **Submission tracking** and reminder emails.

7.7 Market value

Customer	Value	Price
Individual researchers	Faster submission, fewer rejections, better English	$30/month
Labs	Shared templates, co-author workflows	$3k–5k/year
Universities	More publications per faculty; writing-centre replacement	$50k/year
Publishers / journals (B2B)	Pre-submission screening, reviewer assistance	custom

Comparable: Paperpal, Writefull, SciSpace, Trinka — $10–30/month; Paperpal has institutional deals with hundreds of universities.

7.8 How to sell it

1. **"Reviewer 2 simulator"** as a free viral tool: upload an abstract, get a harsh review in 30 seconds.
2. **Writing-centre partnership**: universities run writing support; sell seats through them.
3. **Conference sponsorships & workshops**: "Write your paper in a week with AI agents".
4. **Bundle** with the Literature Review Agent (intro + related work come from Agent 2).
5. **Publisher partnerships**: white-label pre-submission checks.

7.9 Risks & mitigations

Risk	Mitigation
Fabricated citations/results	Only user-provided notes and retrieved references are allowed; a checker flags unsupported claims
Journal AI-use policies	Auto-generate the disclosure statement; keep an edit log
Over-polished but hollow writing	Reviewer simulation catches it; "evidence-per-claim" check
Predatory journals	Blacklist + warning in journal matching

7.10 90-day roadmap

- Weeks 1–2: CLI (this repo) → Weeks 3–6: Word/LaTeX export + live journal data → Weeks 7–10: grammar pass + Zotero → Weeks 11–13: submission tracker & co-author workspace.

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

8 Guide 5 — AI PhD Supervisor Agent

8.1 "Virtual Research Mentor"

8.2 The problem

Many PhD students struggle because:

- supervisors are overloaded and unavailable (one meeting a month is common),
- research direction drifts without anyone noticing,
- there is no continuous, structured feedback,
- students lack methodology and writing coaching,
- the defence arrives with little rehearsal.

Completion rates in many programmes are below 50%, and average time-to-degree keeps growing.

8.3 The solution

An AI **mentor** (assistant to the supervisor, coach to the student) that keeps **long-running memory** of the student's project:

Agent	Role	Output
Research Planning Agent	Creates a 3-year PhD roadmap with milestones, deliverables, risks	`roadmap.md` / `roadmap.json`
Weekly Progress Agent	Student submits a weekly update; agent scores progress vs. plan, flags problems, sets next-week goals	`progress_log.jsonl`, weekly report
Critical Reviewer Agent	Challenges assumptions, methodology and novelty of the current work	`critique.md`
Presentation Agent	Generates conference slides outline, poster text and a mock-defence Q&A	`slides.md`, `poster.md`, `defence_qa.md`

8.3.1 Example

Week 12 update: "Finished the RGB image pipeline; model accuracy 87%; hyperspectral data delayed."

Agent evaluation:

- Progress: 80% of the milestone. On track: Aim 1. At risk: Aim 2 (data delay).
- Problems: "Need more validation experiments — single split, no cross-validation."
- Next week: (1) 5-fold CV, (2) contact core facility for hyperspectral timeline, (3) draft methods section.

8.4 Architecture — memory is the product

```
student profile + topic → Research Planning Agent → roadmap (milestones)
                                                          | stored in memory/
weekly_update.md → Weekly Progress Agent (reads roadmap + last N updates) → evaluation (structured)
                                                          | appended to progress_log.jsonl
      "critique" → Critical Reviewer Agent (reads everything) → critique.md
   "presentation" → Presentation Agent → slides / poster / defence Q&A
```

Memory design

- A **project folder per student** (`memory/<student_slug>/`) with `profile.json`, `roadmap.json`, `progress_log.jsonl`.
- Each call receives a compact **context summary** (roadmap + last 4 updates), not the whole history → cheap and focused.
- Nothing is deleted; the log is the audit trail the supervisor can read.

8.5 Build steps

Step	File	What you learn
1	`memory.py`	Simple file-based memory (JSON/JSONL) with load/append/summarise
2	`agents.py`	Roadmap (structured), weekly evaluation (structured), critic, presentation
3	`main.py`	Sub-commands: `init`, `weekly`, `critique`, `present`

Run:

```
python agent_05_phd_supervisor\main.py init --student "Ali" --topic "Multimodal AI for drought resilience in wheat"
↪  --years 3
python agent_05_phd_supervisor\main.py weekly --student "Ali" --update "Finished RGB pipeline, accuracy 87%, hyperspectral
↪  data delayed"
python agent_05_phd_supervisor\main.py critique --student "Ali"
python agent_05_phd_supervisor\main.py present --student "Ali" --type defence
```

8.6 Product extensions

- **Supervisor dashboard**: all students, traffic-light status, one-click summaries before meetings.
- **Calendar & reminders** (WhatsApp/Email/Slack) for weekly check-ins.
- **Document review**: upload a chapter → structured feedback against the roadmap.
- **Mock viva** with voice; question bank per discipline.
- **Wellbeing signals**: detect stalled progress and suggest supervisor intervention (with consent).
- **Institutional analytics**: cohort completion risk for graduate schools.

8.7 Market value

Customer	Value	Price
PhD students	Weekly structured feedback, faster completion	$10–20/month

Customer	Value	Price
Supervisors	Prepared students, less admin	included in lab plan ($2k–5k/year)
Graduate schools	Higher completion rates, accreditation evidence	$30k–100k/year
Online/remote PhD programmes	Scalable supervision support	custom

The stickiest agent: once a student's 3-year memory lives here, they do not leave.

8.8 How to sell it

1. **Graduate-school pilots**: 20 students for one semester; report completion-risk reduction.
2. **Supervisor champions**: give supervisors a free dashboard; students pay or the department pays.
3. **HEC / accreditation angle** (Pakistan and similar): supervision logs are required evidence — the agent produces them automatically.
4. **Bundle** with the other four agents as the "PhD success plan".
5. **Community**: PhD-student WhatsApp/YouTube communities (Codanics-style) as the distribution channel.

8.9 Risks & mitigations

Risk	Mitigation
Perceived as replacing supervisors	Positioning + supervisor dashboard; supervisor stays in the loop
Over-reliance / wrong advice	Every output carries a "verify with your supervisor" note; critic agent challenges its own advice
Student data privacy	Local/on-prem memory, encryption, opt-in sharing with supervisor
Generic coaching	Memory-grounded prompts; discipline-specific templates

8.10 90-day roadmap

- Weeks 1–2: CLI with file memory (this repo) → Weeks 3–6: web app + supervisor dashboard → Weeks 7–10: chapter review + reminders → Weeks 11–13: mock viva & cohort analytics.

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

9 Guide 6 — The AI Research OS (Platform Vision)

9.1 One platform, five agents, the whole research lifecycle

9.2 What it is

The **AI Research OS** is the orchestrator that connects the five agents into a single, stateful workflow:

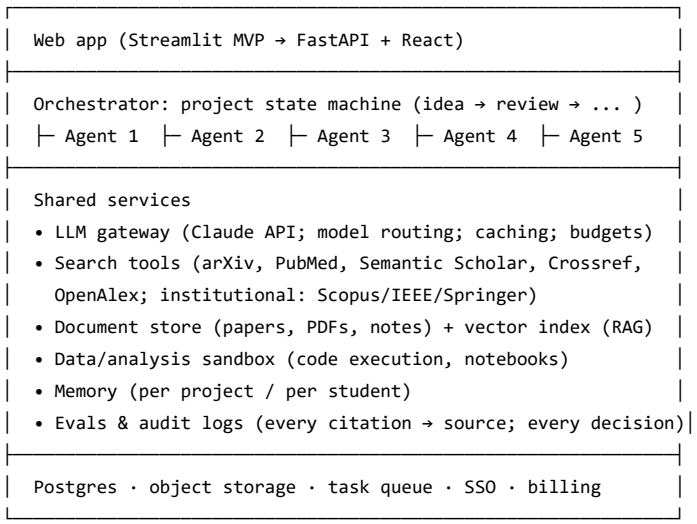
```
Research Idea
      │
      ▼
[1] Research Discovery Agent      → gaps, ideas, proposal
      │
      ▼
[2] Literature Review Agent       → PRISMA review, evidence table, references
      │
      ▼
[3] Experiment Design & Analysis  → design, notebook, figures, stats report
      │
      ▼
[4] Scientific Writing Agent      → manuscript, journal match, reviewer sim, rebuttal
      │
      ▼
[5] Supervisor / Mentor Agent     → roadmap, weekly progress, defence prep
      │
      ▼
   Journal Submission (Phase 3)
```

Positioning: "Cursor + ChatGPT + EndNote + Research Assistant + Bioinformatics Analyst — for scientists."

9.3 Why a platform beats five point tools

Point tools	Platform
Each tool re-asks for context	One **project memory** (topic, papers, data, drafts) shared by all agents
Outputs are files on a laptop	Outputs are **linked objects**: a gap → a review section → an experiment → a results paragraph
Sold seat by seat	Sold as **institutional infrastructure** ($50k–500k/year)
Easy to churn	Sticky: the project history lives inside

9.4 Platform architecture (Phase 2–3)



9.5 Institutional requirements (what universities will ask for)

- **Data residency / on-prem or VPC** deployment.
- **SSO** (SAML/OIDC), role-based access (student, supervisor, admin).
- **Academic-integrity controls**: AI-use disclosure, edit logs, plagiarism check hooks.
- **Usage & outcome reports**: proposals generated, reviews completed, papers submitted.
- **Training & onboarding** (this is where the Codanics content is a competitive advantage).

9.6 Pricing the platform

Tier	Annual price	Seats	Notes
Department	$20k–50k	up to 200	Cloud, shared tenant
University	$50k–150k	unlimited	SSO, reports, training
University + on-prem	$150k–300k	unlimited	Private deployment, support SLA
Enterprise R&D	$200k–500k	custom	Custom pipelines, integrations

Add-ons: training days ($2k–5k/day), custom domain plugins ($20k–100k), evidence-synthesis services.

9.7 Go-to-market for the platform

1. **Land** with one agent (Literature Review) in one department via a free pilot.
2. **Expand** to Discovery + Writing across the faculty (individual champions → dean).
3. **Institutionalise** with SSO, reports and training → university licence.
4. **Upsell** Experiment Agent to labs/core facilities and the Supervisor Agent to the graduate school.
5. **Reference customers** in Pakistan/GCC → regional expansion (South Asia, MENA, Africa, SE Asia).

9.8 Moats

- **Workflow + memory** (not the model): the model is a commodity; the project graph is not.
- **Domain pipelines** (agri, bio, medicine) built with real researchers.
- **Community & training** (Codanics YouTube/WhatsApp/course) → distribution nobody else has in the region.
- **Auditability**: PRISMA trails, citation provenance, reproducible notebooks — what institutions actually buy.

9.9 Team to build it (first 12 months)

Role	Count
Founder / product (research background)	1
AI engineers (agents, tools, evals)	2
Full-stack engineer (web app)	1
Domain scientist / customer success	1
Sales (university & enterprise)	1 (from month 6)

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

10 Guide 7 — Go-to-Market, Pricing & Selling Playbook

10.1 How to actually make money from research agents

10.2 The three customers (and who pays)

Customer	Who uses	Who pays	Sales motion
Individual researcher / PhD student	Themselves	Themselves (card, JazzCash/EasyPaisa, PayPal)	Self-serve, freemium, content marketing
Lab / department	5–50 researchers	PI's grant or department budget	Inside sales, workshops, pilots
Institution / enterprise	Hundreds–thousands	University (ORIC, library, graduate school) or R&D budget	Field sales, RFP/tender, 3–9-month cycle

Rule from the council: *market to individuals, sell to institutions.*

10.3 Pricing (starting point; adjust with data)

Plan	Monthly	Annual	Limits
Free	$0	—	3 discovery runs, 1 review (≤30 papers), watermark
Student	$9 (regional) / $19	$90 / $190	Unlimited discovery, 3 reviews/month, exports
Pro	$29	$290	All five agents, unlimited, priority models
Lab (10 seats)	—	$2,500–5,000	Shared library, admin, invoice billing
Institution	—	$20k–100k	Unlimited seats, SSO, training, reports
Enterprise	—	$100k–500k	On-prem/VPC, custom pipelines, SLA

Regional pricing: Pakistan/South Asia/Africa ≈ 30–40% of US price; GCC ≈ 80–100%.

Cost side (rough): a typical literature review of 80 papers costs a few dollars in LLM tokens with Claude Opus 5 (adaptive thinking, structured outputs, prompt caching) — margins are healthy at $29/month if you cap heavy usage or route bulk screening to a cheaper model.

10.4 Channels

1. **Content (Codanics engine)** — YouTube tutorials & demos, WhatsApp channel, the free course playlist, the paid course. Every video ends with "try the agent free".
2. **Community** — PhD-student groups, ResearchGate, LinkedIn, X/Twitter academic communities, Discord.
3. **Universities** — ORIC directors, libraries, graduate schools, deans of research. Offer seminars + pilots.
4. **Conferences & journals clubs** — live demos; sponsor student sessions.
5. **Partners** — reference-manager tools, publishers, core facilities, EdTech resellers in GCC.

10.5 The pilot playbook (institutions)

Step	Action	Artefact
1	Find a champion (a supervisor or ORIC officer)	Discovery call notes
2	3-month free pilot for one department (≤50 users)	Pilot agreement (1 page)
3	Onboarding workshop (2 hours)	Slides + recording
4	Weekly usage & outcome report	Dashboard/email
5	End-of-pilot report: time saved, reviews completed, proposals generated	PDF report
6	Proposal to dean/ORIC with 3 tiers	Quote + contract

Success metrics that convince deans: **time-to-literature-review ↓ 60%**, **proposals submitted ↑**, **papers submitted ↑**, **supervision logs available for accreditation**.

10.6 Sales collateral you need

- One-page product sheet per agent (use the guides in this folder).
- A 10-minute demo video per agent.
- ROI calculator (hours saved × researcher hourly cost).
- Security & privacy sheet (data flow, storage, LLM provider, retention).
- Academic-integrity statement (how outputs are traceable; AI-use disclosure).
- Case study from the first pilot.

10.7 Objections & answers

Objection	Answer
"AI hallucinates citations."	Every citation comes from a retrieved record with DOI/ID; the model cannot cite outside the retrieved set; we show provenance.
"Journals ban AI writing."	Most allow AI assistance with disclosure; we generate the disclosure and keep an edit log; the human remains the author.
"Our data is sensitive."	On-prem/VPC option; only summaries go to the LLM for the Experiment Agent; regional hosting.

Objection	Answer
"Students will cheat."	The agent is a coach with an audit trail; supervisors see logs; it improves — not replaces — student work.
"We already have Scopus/Turnitin."	Complementary: we sit on top of databases and automate the workflow; integrations available.

10.8 12-month revenue model (illustrative)

Quarter	Individuals (paying)	Labs	Institutions	ARR
Q1	100 × $19	2	0	~$30k
Q2	400	6	1 pilot → 1 paid ($30k)	~$150k
Q3	900	12	2 ($40k avg)	~$330k
Q4	1,500	20	4 ($50k avg)	~$600k

10.9 Legal & compliance checklist

- Terms of service + acceptable-use (no fabrication, disclosure required).
- Privacy policy (GDPR-style; local data-protection laws).
- LLM provider data-processing terms (no training on customer data).
- Academic-integrity guidance document for institutions.
- Invoicing & tax (local company + international payments).

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

Part III

Part III — Learning to build AI agents

11 What to Learn to Build AI Agents (Roadmap)

A practical, ordered learning path — the same order used in **Python ka Chilla: Build AI Agents** by Codanics. Free playlist: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

11.1 Stage 0 — Python you actually need (2–3 weeks)

Topic	Why it matters for agents	Used in this repo
Functions, modules, packages	Every tool is a function; every agent is a module	all `tools.py`, `agents.py`
Type hints & dataclasses / Pydantic	Structured outputs, validation	every `BaseModel`
`pathlib`, JSON, CSV	Saving outputs, memory, audit trails	`main.py`, `memory.py`
Virtual environments (`venv`), `pip`, `requirements.txt`	Reproducible installs	`02_agents_code/.venv`
Environment variables & `.env`	Secrets (API keys) never in code	`llm.py` (`python-dotenv`)
Exceptions & retries	APIs fail; agents must not crash	`tools.search_all`
HTTP with `requests`	Calling research databases	`tools.py`
CLI with `typer`, pretty output with `rich`	Usable tools without a UI	every `main.py`
pandas / numpy basics	Data agents	Agent 3

11.2 Stage 1 — LLM fundamentals (1–2 weeks)

1. **How an LLM call works**: system prompt, user message, tokens, `max_tokens`, temperature vs. effort, streaming.
2. **Prompting**: role, task, constraints, examples, output format. Learn to write *system prompts* that define a persona and rules (see `CITE_RULE` in `agents.py`).
3. **Structured outputs**: force JSON that matches a schema (Pydantic → JSON schema). This is the single most important skill for reliable agents.
4. **Context windows & cost**: what to put in the prompt (only what is needed), prompt caching, choosing models (`claude-opus-5` for quality, `claude-sonnet-5` for bulk).
5. **Thinking/effort controls**: adaptive thinking; low effort for screening, high for critique.

11.3 Stage 2 — Tools & agents (2–3 weeks)

1. **Tool use / function calling**: give the model functions it can call; the tool-runner loop (call → execute → feed result back).
2. **The agent loop**: *Plan → Act (tool) → Observe → Repeat → Answer* (ReAct).
3. **Pipelines vs. autonomous agents**: this repo uses *pipelines of small agents* (predictable, testable). Fully autonomous loops come later.
4. **Multi-agent patterns**: planner/worker, generator/critic, persona simulation (Agent 4), orchestrator (`main.py`).
5. **Memory**: short-term (the prompt), long-term (files/DB), summaries (`context_summary()` in Agent 5).
6. **Retrieval (RAG)**: search → chunk → embed → retrieve → answer; citation grounding.

11.4 Stage 3 — Research-domain tools (1–2 weeks)

- Scholarly APIs: arXiv, PubMed E-utilities, Semantic Scholar, Crossref, OpenAlex.
- PDF text extraction (`pypdf`, GROBID), reference formats (BibTeX/RIS), DOIs.
- Systematic-review methodology (PRISMA), evidence tables, PICO.
- Basic statistics: t-tests, ANOVA, multiple-testing (FDR), effect sizes, cross-validation, leakage.

11.5 Stage 4 — Productising (3–4 weeks)

1. **Evaluation**: golden sets, rubric grading, LLM-as-judge, regression tests for prompts.
2. **Web app**: Streamlit (MVP) → FastAPI + React; background jobs; file uploads.
3. **Storage**: SQLite/Postgres for papers, runs and memory; object storage for PDFs.
4. **Observability**: logs of every prompt/response, token usage, cost per run.
5. **Safety & integrity**: no citation without a source; AI-use disclosures; privacy (send summaries not raw data).
6. **Deployment**: Docker, cloud VM/containers, scheduled runs, secrets management.
7. **Business**: pricing, pilots, sales collateral (see `01_guides/07_go_to_market_and_pricing.md`).

11.6 Stage 5 — Advanced (ongoing)

- Autonomous tool-runner agents with approval gates; sandboxed code execution.
- Managed / hosted agents with scheduled deployments; MCP servers for tools.
- Fine-tuning vs. prompting; embeddings & vector DBs; graph-based citation analysis.
- Multi-modal inputs (figures, PDFs, images) for extraction.

11.7 Suggested weekly plan (8 weeks)

Week	Build
1	Python refresh; run `agent_03 --demo --no-llm` (no key needed)
2	Get an API key; run Agent 1 with `--offline`, then live
3	Modify `tools.py`: add OpenAlex; add a new field to `Paper`
4	Run Agent 2; edit `evidence.csv`; `--from-evidence`
5	Run Agent 3 on your own CSV; add a new test in `analysis.py`
6	Run Agent 4 on your notes; add a 4th reviewer persona
7	Run Agent 5 for 3 "weeks"; build a Streamlit status page
8	Write evals; pick one agent; make it a product (UI + pricing page)

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

12 Glossary — Major Terms in AI Agent Building

Short, practical definitions with a pointer to where each concept appears in this repo.

12.1 Core LLM terms

Term	Meaning	In this repo
LLM (Large Language Model)	A neural network that predicts text; used here via the Claude API.	`llm.py`
Prompt	The text sent to the model: *system* (role & rules) + *user* (task & data).	every agent's `SYSTEM`
System prompt	Persistent instructions defining persona, rules and output format.	`CITE_RULE`, `NO_FABRICATION`
Token	The unit of text the model reads/writes (~¾ of a word). Cost and limits are in tokens.	`max_tokens` in `llm.py`
Context window	Maximum tokens the model can consider at once (1M for Claude Opus 5).	`context_summary()` keeps prompts small
Temperature / sampling	Randomness control (removed on current Claude models — use *effort* instead).	—
Effort	How much reasoning the model spends (`low`…`max`). Cheaper for simple tasks.	`effort=` per call
Adaptive / extended thinking	The model reasons privately before answering.	`thinking={"type": "adaptive"}`
Streaming	Receive output token-by-token; avoids timeouts on long outputs.	`messages.stream(...)`
Structured output	Forcing the answer to match a JSON schema.	`ask_structured()`
Pydantic model	A Python class that validates data and generates the JSON schema.	`class Gap(BaseModel)`
Hallucination	Fluent but false output (e.g. invented citations).	mitigated by tool-grounded citation rules
Grounding	Tying model output to retrieved facts (papers, numbers).	`Paper.short()` blocks in prompts
Prompt caching	Re-using a stable prompt prefix to cut cost/latency.	recommended for web app
Refusal / stop reason	Why generation stopped (`end_turn`, `max_tokens`, `refusal`).	`_check()` in `llm.py`

12.2 Agent terms

Term	Meaning	In this repo
Agent	An LLM that pursues a goal by taking actions (tool calls) and observing results.	all five
Tool / function calling	Letting the model call code (search, calculate, save).	`tools.py`
Tool runner / agent loop	The loop: model proposes tool call → code runs it → result goes back → repeat.	pattern described in `03_how_these_agents_work.md`
ReAct	"Reason + Act": interleave reasoning with tool actions.	conceptual
Pipeline (workflow)	A fixed sequence of LLM steps controlled by code. Predictable and testable.	every `main.py`
Orchestrator	Code (or an LLM) that decides which agent runs next.	`main.py`; Research OS vision
Planner / worker	One agent decomposes the task; others execute parts.	`LiteratureMiningAgent.plan`
Critic / reviewer	An agent that checks another's output.	`StatisticsAgent`, `ReviewerSimulationAgent`
Persona	A role the model plays (methodologist, domain expert).	`PERSONAS` in Agent 4
Multi-agent system	Several specialised agents cooperating.	Agents 1–5
Human-in-the-loop	A person reviews/edits between steps.	`--from-evidence` in Agent 2
Memory (short/long-term)	Prompt context vs. persisted state across runs.	`memory.py`
State	Everything the agent knows at a point in time (files, DB).	`outputs/`, `memory/`
Guardrails	Rules/validators that constrain behaviour.	`CITE_RULE`, strict schemas
Evaluation (evals)	Tests measuring agent quality on known cases.	Stage 4 of roadmap
LLM-as-judge	Using a model to grade outputs against a rubric.	evals
RAG (Retrieval-Augmented Generation)	Retrieve relevant documents, then generate with them in the prompt.	search → prompt in Agents 1–2
Embedding / vector DB	Numeric representations for semantic search.	extension idea
MCP (Model Context Protocol)	A standard for exposing tools/data to models.	extension idea
Managed / hosted agents	Provider runs the agent loop and sandbox for you.	extension idea
Sandbox / code execution	Safe environment where an agent runs generated code.	Agent 3 extension
Idempotent run	Running twice with the same input produces the same files.	`outputs/<slug>/`

12.3 Research-domain terms

Term	Meaning
DOI / arXiv ID / PMID	Persistent identifiers for papers; our citation keys.
PRISMA	Reporting standard for systematic reviews (identified → screened → included).
PICO	Population, Intervention, Comparison, Outcome — frames a review question.
Inclusion / exclusion criteria	Rules deciding which papers enter a review.
Evidence table	Structured extraction: study, dataset, method, result, limitations.
Research gap	A specific unanswered question or missing combination in the literature.
Novelty statement	One paragraph explaining what is new and why it matters.
Hypothesis / research question	Testable statement / question the study addresses.
Replicates, controls, covariates	Design elements that make experiments valid.
QC / normalisation	Cleaning and scaling data before analysis.
Differential analysis	Testing which features differ between groups.
p-value / FDR (Benjamini–Hochberg)	Significance and multiple-testing correction.
Cross-validation / leakage	Honest model evaluation; leakage = test info entering training.
IMRaD	Introduction, Methods, Results, and Discussion — manuscript structure.
Impact factor / APC / open access	Journal metrics and publishing costs.
Peer review / rebuttal	Expert evaluation and the authors' point-by-point response.
Viva / defence	Oral examination of a thesis.

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

13 How These AI Agents Work (Architecture & Patterns)

The mental model behind all five agents, explained once. Read this before reading the code.

13.1 The one-sentence model

An agent = a model + tools + a loop + state.

- **Model**: Claude (via `llm.py`) — reasons, plans, writes.
- **Tools**: Python functions the model's output triggers or that feed the model (search APIs, statistics, memory).
- **Loop**: the control flow — in this repo, mostly *code-controlled pipelines* (predictable) rather than open-ended autonomous loops.
- **State**: what persists — `outputs/` files, `memory/` folders, JSON/CSV.

13.2 Two ways to run a loop

13.2.1 a) Code-controlled pipeline (used in this repo)

```
main.py:  step1 = AgentA.run(input)        # LLM call
          data  = tool(step1)              # Python
          step2 = AgentB.run(data)         # LLM call (structured)
          ...
```

Pros: deterministic order, easy to test, easy to explain to a customer, cheap. Cons: less flexible when the task is open-ended.

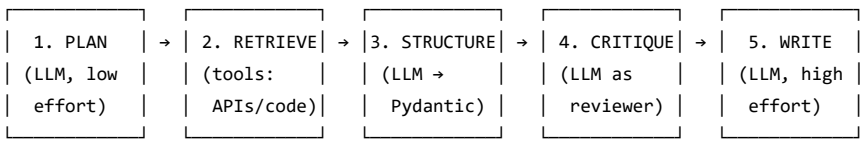
13.2.2 b) Model-controlled tool loop (ReAct / tool runner)

```
messages = [user goal]
loop:
   response = model(messages, tools=[search, analyse, save])
   if response has tool calls:  run them, append results, continue
   else: done
```

Pros: the model decides what to do next (great for exploration). Cons: harder to predict cost and behaviour; needs guardrails and evals.

Our rule: start with (a); introduce (b) inside a single step when the step is genuinely open-ended (e.g. "search until you have 50 relevant papers").

13.3 The five-step template every agent follows



Agent	Plan	Retrieve	Structure	Critique	Write
1 Discovery	search queries	4 databases	trends, gaps, ideas	gap agent challenges the field	proposal
2 Lit Review	protocol	4 databases	screening, evidence rows	(dual screening — extension)	review
3 Experiment	design	analysis engine	design, stats review	statistics agent	methods/results
4 Writing	—	notes + journal catalogue	journal ranking, reviews	reviewer simulation	manuscript, rebuttal
5 Supervisor	roadmap	memory	weekly evaluation	critical reviewer	slides/defence

13.4 Structured outputs — the backbone

```python
class Decision(BaseModel):
    paper_id: str
    decision: Literal["include", "exclude", "maybe"]
    reason: str

res = ask_structured(SYSTEM, prompt, ScreeningBatch)   # guaranteed to parse
```

Why it matters: - Outputs become **data** (tables, JSON, DB rows) rather than prose to parse with regex. - You can **validate** (`ge=1, le=10` scores; enums) and **test**. - Downstream agents receive **clean inputs** (`gaps.model_dump_json()`).

13.5 Grounding & anti-hallucination

1. **Tools own the facts.** Papers come from arXiv/PubMed/etc.; numbers come from pandas/scipy.
2. **The prompt contains the allowed facts** (`Paper.short()` blocks, `summary.json`).
3. **The system prompt forbids anything else** (`CITE_RULE`, `NO_FABRICATION`).
4. **Code checks**: PRISMA counts computed in Python; missing screening decisions default to `maybe`.
5. **Human-in-the-loop**: editable CSVs; `--from-evidence` re-runs.

13.6 Memory

Kind	Where	Example
Working memory	the prompt	the corpus of abstracts in one call
Run memory	`outputs/<slug>/`	`papers.json` reused by later steps
Long-term memory	`memory/<student>/`	roadmap + JSONL progress log

Kind	Where	Example
Summarised memory	`context_summary()`	roadmap + last 4 updates, not the full log

Rule: **send the model a summary, keep the full record on disk.**

13.7 Multi-agent patterns you will see

- **Planner → Workers**: `LiteratureMiningAgent.plan()` produces queries; tools execute them.
- **Generator → Critic → Reviser**: `ManuscriptAgent` → `ReviewerSimulationAgent` → `RevisionAgent`.
- **Persona ensemble**: three reviewers with different priorities → more diverse critique.
- **Batching**: screen 10 abstracts per call for cost and reliability.
- **Orchestrator**: `main.py` today; the *AI Research OS* tomorrow.

13.8 Cost & quality levers

Lever	Effect
`effort="low"` for screening/planning	3–5× cheaper, same accuracy on easy tasks
Smaller model (`claude-sonnet-5`) for bulk	cheaper; keep Opus for gaps/critique/writing
Shorter abstracts (`Paper.short(900)`)	fewer input tokens
Prompt caching on stable prefixes	up to 90% cheaper repeated context
Streaming	no timeouts on long writes
Batches (async)	50% cheaper for non-urgent bulk screening

13.9 Evaluation — how you know it works

1. **Golden topics**: 5 topics where you know the key papers → measure recall of Agent 1/2 search.
2. **Screening agreement**: compare agent decisions with a human on 50 abstracts (Cohen's kappa).
3. **Extraction accuracy**: spot-check 20 evidence rows against abstracts.
4. **Rubric grading**: LLM-as-judge scores proposals/reviews on specificity, grounding, structure.
5. **Regression**: rerun after any prompt change; diff the JSON.

13.10 From CLI to product

```
CLI (this repo) → Streamlit app → FastAPI + workers → multi-tenant SaaS → on-prem edition
       ↑ evals & logging from day one ↑
```

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

14 Resources & Community

14.1 Learn with Codanics (start here)

Resource	Link
🎓 Paid deep-dive course — **Python ka Chilla: Build AI Agents**	https://codanics.com/courses/python-ka-chilla-build-ai-agents/
🆓 Complete free course playlist (YouTube)	https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40
▶️ YouTube channel	@codanics
💬 WhatsApp channel (announcements, Q&A)	https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
🌐 Website	https://codanics.com

14.2 APIs used in the code (all free tiers)

Service	Docs
Anthropic Claude API (Python SDK `anthropic`)	https://docs.anthropic.com/
arXiv API (`arxiv` package)	https://info.arxiv.org/help/api/
Semantic Scholar Graph API	https://api.semanticscholar.org/
PubMed E-utilities	https://www.ncbi.nlm.nih.gov/books/NBK25501/
Crossref REST API	https://api.crossref.org/
OpenAlex (recommended next source)	https://docs.openalex.org/

14.3 Python libraries used

`anthropic`, `pydantic`, `python-dotenv`, `requests`, `arxiv`, `typer`, `rich`, `python-slugify`, `pandas`, `numpy`, `scipy`, `scikit-learn`, `statsmodels`, `matplotlib`, `nbformat`, `pypdf`.

14.4 Methodology references

- PRISMA 2020 statement — https://www.prisma-statement.org/
- Cochrane Handbook for Systematic Reviews — https://training.cochrane.org/handbook
- Kitchenham, B. — Guidelines for performing Systematic Literature Reviews in Software Engineering
- ICMJE recommendations on authorship & AI use — https://www.icmje.org/
- COPE guidance on AI in research publishing — https://publicationethics.org/

14.5 Comparable products (study their pricing & UX)

Elicit · Consensus · Scite · SciSpace · Research Rabbit · Litmaps · Covidence · Rayyan · DistillerSR · Paperpal · Writefull · Trinka.

14.6 Tools for the next steps

- **Streamlit** (MVP UI) — https://streamlit.io/
- **FastAPI** (backend) — https://fastapi.tiangolo.com/
- **Quarto** (this book) — https://quarto.org/
- **Docker** (deployment) — https://docs.docker.com/

Codanics — Python ka Chilla: Build AI Agents

- Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/
- Website: https://codanics.com · YouTube: @codanics
- WhatsApp channel: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r
- Complete free course: https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40

Part IV

Part IV — The code

15 The code — setup and shared pattern

All five agents live in `02_agents_code/`, share one `.venv` and follow the same file pattern (`llm.py`, `tools.py`, `agents.py`, `main.py`). This chapter shows the setup files; the following chapters list every source file of each agent.

15.1 `02_agents_code/README.md`

15.1.1 `README.md`

````
# 02_agents_code — Five independent AI research agents (Python)

One shared virtual environment (`.venv`), one `.env` with your API key, five **self-contained** agent folders.
No agent imports from another agent — copy any folder out and it still runs.

```
02_agents_code/
├── .venv/                          ← shared virtual environment (created by setup script)
├── requirements.txt                ← all dependencies for all five agents
├── .env.example → .env             ← ANTHROPIC_API_KEY, CLAUDE_MODEL, optional search keys
├── setup_venv.ps1 / setup_venv.sh  ← one-shot setup
├── agent_01_research_discovery/    ← topic → papers → gaps → ideas → proposal
├── agent_02_literature_review/     ← question → PRISMA search/screen/extract → review
├── agent_03_research_experiment/   ← design → analysis (code) → stats review → methods + notebook
├── agent_04_academic_writing/      ← notes → manuscript → journal match → reviewer sim → rebuttal
└── agent_05_phd_supervisor/        ← roadmap → weekly progress → critique → presentation (memory)
```

## Setup (Windows PowerShell)

```powershell
cd 02_agents_code
.\setup_venv.ps1                      # creates .venv, installs requirements, copies .env.example → .env
notepad .env                           # paste your ANTHROPIC_API_KEY
.\.venv\Scripts\Activate.ps1
```

macOS / Linux: `bash setup_venv.sh && source .venv/bin/activate`

## Run each agent

```powershell
python agent_01_research_discovery\main.py --topic "AI-based drought prediction in wheat"
python agent_02_literature_review\main.py --question "AI methods for plant disease detection"
python agent_03_research_experiment\main.py --description "RNA-seq of heat-stressed rice, control vs heat, n=3" --demo
python agent_04_academic_writing\main.py --notes agent_04_academic_writing\sample_notes.md --field "plant science"
python agent_05_phd_supervisor\main.py init --student "Ali" --topic "Multimodal AI for drought resilience in wheat"
```

Add `--help` to any command to see all options. Agents 1 and 2 accept `--offline` (built-in demo papers); Agent 3 accepts
↪ `--no-llm` (numeric pipeline only, no API key needed).
````

```
## The shared pattern inside every agent

| File | Role |
|------|------|
| `llm.py` | `ask()` — streamed free text; `ask_structured()` — JSON constrained to a Pydantic model. Model = `CLAUDE_
↪  MODEL` env var (default `claude-opus-5`), adaptive thinking, effort per call. |
| `tools.py` | Deterministic functions the agent calls (search APIs, data analysis, memory) |
| `agents.py` | Small single-responsibility agents: a system prompt + one call + a Pydantic output model |
| `main.py` | A Typer CLI that orchestrates the agents and saves everything to `outputs/` |

### Why this design

- **Structured outputs** make agents composable and testable — every decision is data, not prose.
- **Tools own the facts** (papers, numbers); the model owns reasoning and writing → fewer hallucinations.
- **Streaming** avoids timeouts on long outputs (reviews, manuscripts).
- **One file to swap providers**: only `llm.py` knows about the API.

### Cost tips

- Set `CLAUDE_MODEL=claude-sonnet-5` in `.env` for cheap bulk runs (screening 500 abstracts), keep `claude-opus-5` for
↪  writing and critique.
- `effort` is set per call (`low` for screening/planning, `high` for gaps/critique/writing).
- Enable prompt caching on the big stable prefixes (protocol + corpus) when you move to a web app.

---

**Codanics — Python ka Chilla: Build AI Agents**
Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/ · Website: https://codanics.com · YouTube:
↪  [@codanics](https://www.youtube.com/@codanics)
WhatsApp: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r · Free course:
↪  https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40
```

15.2 requirements.txt

15.2.1 `requirements.txt`

```
# Codanics - Python ka Chilla: Build AI Agents
# https://codanics.com/courses/python-ka-chilla-build-ai-agents/
#
# One shared virtual environment for all five agents.
#   python -m venv .venv
#   .\.venv\Scripts\Activate.ps1   (Windows)   |   source .venv/bin/activate (macOS/Linux)
#   pip install -r requirements.txt

# --- LLM + structured outputs ---
anthropic
pydantic>=2.7
python-dotenv

# --- research search tools (free public APIs) ---
requests
arxiv

# --- CLI / console ---
```

```
typer
rich
python-slugify

# --- data, statistics, ML (Experiment Agent) ---
pandas
numpy
scipy
scikit-learn
statsmodels
matplotlib
nbformat
tabulate

# --- documents ---
pypdf
```

15.3 .env.example

15.3.1 .env.example

```
# Copy this file to .env and fill in your keys.  Never commit .env.

# Required: Anthropic Claude API key  (https://console.anthropic.com/)
ANTHROPIC_API_KEY=sk-ant-...

# Optional: model override (default is claude-opus-5). e.g. claude-sonnet-5 for cheaper bulk runs
CLAUDE_MODEL=claude-opus-5

# Recommended: free Semantic Scholar API key - anonymous access is often rate-limited (HTTP 429)
# https://www.semanticscholar.org/product/api
SEMANTIC_SCHOLAR_API_KEY=

# Optional but polite for PubMed E-utilities (NCBI asks for an email; API key raises rate limit)
NCBI_EMAIL=you@example.com
NCBI_API_KEY=
```

15.4 setup_venv.ps1

15.4.1 setup_venv.ps1

```
# Codanics - Python ka Chilla: Build AI Agents
# Creates the shared virtual environment and installs all dependencies (Windows PowerShell).
#   PS> .\setup_venv.ps1
python -m venv .venv
.\.venv\Scripts\Activate.ps1
python -m pip install --upgrade pip
pip install -r requirements.txt
if (-not (Test-Path .env)) { Copy-Item .env.example .env; Write-Host "Created .env - add your ANTHROPIC_API_KEY" }
Write-Host "Done. Try: python agent_01_research_discovery\main.py --topic 'AI-based drought prediction in wheat'"
```

15.5 setup_venv.sh

15.5.1 setup_venv.sh

```
#!/usr/bin/env bash
# Codanics - Python ka Chilla: Build AI Agents
# Creates the shared virtual environment and installs all dependencies (macOS / Linux).
#   $ bash setup_venv.sh
set -e
python3 -m venv .venv
source .venv/bin/activate
python -m pip install --upgrade pip
pip install -r requirements.txt
[ -f .env ] || { cp .env.example .env; echo "Created .env - add your ANTHROPIC_API_KEY"; }
echo "Done. Try: python agent_01_research_discovery/main.py --topic 'AI-based drought prediction in wheat'"
```

15.6 .gitignore

15.6.1 .gitignore

```
.venv/
.env
__pycache__/
*.pyc
**/outputs/
**/memory/
*.ipynb_checkpoints
```

16 Code: Agent 1 — Research Discovery

Folder: `02_agents_code/agent_01_research_discovery/` — self-contained; run `python agent_01_research_discovery/main.py --help`.

16.1 README

16.1.1 `README.md`

````
# Agent 1 — AI Research Discovery Agent ("Your AI Research Scientist")

Turns a one-line research interest into: retrieved papers → trend report → research gaps → ranked ideas → a research
↪  proposal.

```
topic ─► LiteratureMiningAgent (plan queries, analyse trends)
      ─► search tools: arXiv · Semantic Scholar · PubMed · Crossref   (real, free APIs)
      ─► ResearchGapAgent  (structured gaps with evidence IDs)
      ─► IdeaGeneratorAgent (5 ranked ideas: novelty / feasibility / impact)
      ─► ProposalWriterAgent (proposal.md)
```

## Files

| File | Purpose |
|------|---------|
| `main.py` | CLI orchestrator (`--topic`, `--max-papers`, `--sources`, `--offline`) |
| `agents.py` | The four sub-agents + their Pydantic output models |
| `tools.py` | Search functions for 4 databases, a common `Paper` model, de-duplication, offline sample |
| `llm.py` | Thin Claude client: `ask()` (streamed text) and `ask_structured()` (JSON → Pydantic) |
| `outputs/` | One folder per topic: `papers.json`, `trends.json`, `gaps.json`, `ideas.json`, `proposal.md` |

## Run

```powershell
# from 02_agents_code with the .venv activated and ANTHROPIC_API_KEY in .env
python agent_01_research_discovery\main.py --topic "AI-based drought prediction in wheat"
python agent_01_research_discovery\main.py --topic "..." --max-papers 60 --sources arxiv,semantic_scholar
python agent_01_research_discovery\main.py --topic "..." --offline      # no internet needed for search
```

## What this agent teaches

1. **Tool use with real APIs** — normalising four different APIs into one `Paper` model.
2. **Structured outputs** — `ask_structured()` forces JSON that matches a Pydantic model.
3. **Citation grounding** — the model may only cite `[ids]` present in its input.
4. **Pipeline orchestration** — small agents, each with one job, chained in `main.py`.

## Ideas to extend
````

```
- Weekly scheduled runs + "what's new" diff → email digest.
- Citation network (Semantic Scholar references/citations) + `networkx` centrality.
- Streamlit UI; save papers to SQLite; RAG-style Q&A over abstracts.

---

**Codanics — Python ka Chilla: Build AI Agents**
Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/ · Website: https://codanics.com · YouTube:
↪ [@codanics](https://www.youtube.com/@codanics)
WhatsApp: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r · Free course:
↪ https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40
```

16.2 Source files

16.2.1 `agents.py`

```
"""
agents.py — the four sub-agents of the Research Discovery Agent.

    1. LiteratureMiningAgent  : query expansion + trend / key-author analysis
    2. ResearchGapAgent       : explicit, evidence-linked gap statements
    3. IdeaGeneratorAgent     : ranked research / PhD topic ideas
    4. ProposalWriterAgent    : a research proposal for the best idea

Each agent = one system prompt + one (structured or free-text) model call.
This "small, single-responsibility agents" style is easy to test and easy to teach.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from pydantic import BaseModel, Field

from llm import ask, ask_structured
from tools import Paper

BRAND = ("Codanics — Python ka Chilla: Build AI Agents | "
         "https://codanics.com/courses/python-ka-chilla-build-ai-agents/")

CITE_RULE = (
    "CITATION RULE: You may cite ONLY the papers listed in the input, using their exact [id] tags "
    "(e.g. [arxiv:2401.01234], [pmid:38012345], [doi:10.1000/xyz]). Never invent papers, authors or IDs."
)


# ============================================================ 1. Literature Mining
class SearchPlan(BaseModel):
    queries: list[str] = Field(description="3-6 database search queries (boolean/keyword style)")
    key_concepts: list[str]


class TrendReport(BaseModel):
    summary: str = Field(description="4-6 sentence overview of the field based on the papers")
```

```
    themes: list[str] = Field(description="Main research themes with cited [ids]")
    rising_methods: list[str] = Field(description="Methods/technologies gaining traction, with [ids]")
    key_authors_or_groups: list[str]
    data_sources_used: list[str] = Field(description="Datasets / data types used in the literature")


class LiteratureMiningAgent:
    SYSTEM = f"You are a senior research librarian and scientometrics expert. {CITE_RULE}"

    def plan(self, topic: str) -> SearchPlan:
        return ask_structured(
            self.SYSTEM,
            f"Research interest: {topic}\n\nDesign search queries that maximise recall across arXiv, "
            f"PubMed, Semantic Scholar and Crossref. Include synonyms and adjacent methods.",
            SearchPlan, effort="low")

    def analyse(self, topic: str, papers: list[Paper]) -> TrendReport:
        corpus = "\n".join(p.short() for p in papers)
        return ask_structured(
            self.SYSTEM,
            f"Topic: {topic}\n\nPapers:\n{corpus}\n\nAnalyse themes, rising methods, key authors and data sources.",
            TrendReport)


# ============================================================ 2. Research Gap
class Gap(BaseModel):
    statement: str = Field(description="One precise gap statement, e.g. 'Existing studies predict drought "
                                       "stress using RGB imaging, but few integrate hyperspectral + genomic data.'")
    evidence_ids: list[str] = Field(description="Paper [ids] that support this gap")
    why_it_matters: str
    difficulty: str = Field(description="low | medium | high")


class GapAnalysis(BaseModel):
    gaps: list[Gap]


class ResearchGapAgent:
    SYSTEM = (f"You are a critical research strategist who finds what has NOT been done. {CITE_RULE} "
              "Gaps must be specific (data, method, population, scale, validation), not generic.")

    def run(self, topic: str, papers: list[Paper], trends: TrendReport) -> GapAnalysis:
        corpus = "\n".join(p.short() for p in papers)
        return ask_structured(
            self.SYSTEM,
            f"Topic: {topic}\n\nTrend report:\n{trends.model_dump_json(indent=1)}\n\nPapers:\n{corpus}\n\n"
            f"Identify 5-8 concrete research gaps.",
            GapAnalysis, effort="high")


# ============================================================ 3. Idea Generation
class Idea(BaseModel):
    title: str
    one_paragraph: str
    addresses_gaps: list[int] = Field(description="Indices (0-based) of the gaps this idea addresses")
    novelty_score: int = Field(ge=1, le=10)
    feasibility_score: int = Field(ge=1, le=10)
    impact_score: int = Field(ge=1, le=10)
    required_data: list[str]
```

```
    required_skills: list[str]
    suitable_for: str = Field(description="MS thesis | PhD | grant | industry R&D")


class IdeaList(BaseModel):
    ideas: list[Idea]


class IdeaGeneratorAgent:
    SYSTEM = ("You are an experienced PhD supervisor and grant reviewer. Propose research topics that are "
              "novel, feasible in 2-4 years and clearly linked to the identified gaps. " + CITE_RULE)

    def run(self, topic: str, gaps: GapAnalysis) -> IdeaList:
        return ask_structured(
            self.SYSTEM,
            f"Topic: {topic}\n\nGaps:\n{gaps.model_dump_json(indent=1)}\n\nPropose 5 ranked ideas (best first).",
            IdeaList, effort="high")


# ============================================================ 4. Proposal Writer
class ProposalWriterAgent:
    SYSTEM = ("You write concise, rigorous research proposals in Markdown for university review committees. "
              + CITE_RULE + " Use headings: Title, Background, Novelty Statement, Research Questions, "
              "Hypotheses, Objectives, Methodology, Expected Outcomes, Timeline (Gantt-style table), "
              "Risks & Mitigations, References (only cited ids).")

    def run(self, topic: str, idea: Idea, gaps: GapAnalysis, papers: list[Paper], echo: bool = False) -> str:
        corpus = "\n".join(p.short(400) for p in papers)
        body = ask(
            self.SYSTEM,
            f"Research interest: {topic}\n\nSelected idea:\n{idea.model_dump_json(indent=1)}\n\n"
            f"Gaps:\n{gaps.model_dump_json(indent=1)}\n\nPapers available for citation:\n{corpus}\n\n"
            f"Write a 2-3 page proposal.",
            effort="high", echo=echo)
        return body + f"\n\n---\n*Generated with the AI Research Discovery Agent — {BRAND}*\n"
```

16.2.2 `llm.py`

```
"""
llm.py — a thin wrapper around the Anthropic Claude API used by this agent.

Two functions are all you need:
    ask(system, user)                  -> free text (streamed)
    ask_structured(system, user, Model) -> a validated Pydantic object (JSON structured output)

Why a wrapper?  Every agent in this project talks to the model the same way.  Keeping the
API details in ONE file per agent means you can swap the model or provider in one place.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import os
from typing import Type, TypeVar
```

```python
import anthropic
from dotenv import load_dotenv
from pydantic import BaseModel

load_dotenv()  # reads ANTHROPIC_API_KEY / CLAUDE_MODEL from a .env file if present

MODEL = os.getenv("CLAUDE_MODEL", "claude-opus-5")
T = TypeVar("T", bound=BaseModel)

_client: anthropic.Anthropic | None = None


def client() -> anthropic.Anthropic:
    """Lazily create one shared client (credentials come from the environment)."""
    global _client
    if _client is None:
        _client = anthropic.Anthropic()
    return _client


def _check(msg: anthropic.types.Message) -> None:
    if msg.stop_reason == "refusal":
        detail = getattr(msg, "stop_details", None)
        raise RuntimeError(f"The model refused this request: {detail}")
    if msg.stop_reason == "max_tokens":
        raise RuntimeError("Output was cut off (max_tokens). Increase max_tokens and retry.")


def _text(msg: anthropic.types.Message) -> str:
    return "".join(b.text for b in msg.content if b.type == "text")


def ask(system: str, user: str, *, max_tokens: int = 16000, effort: str = "medium",
        echo: bool = False) -> str:
    """Free-text generation. Streams so long outputs never hit HTTP timeouts."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
    ) as stream:
        if echo:
            for chunk in stream.text_stream:
                print(chunk, end="", flush=True)
            print()
        msg = stream.get_final_message()
    _check(msg)
    return _text(msg)


def ask_structured(system: str, user: str, model_cls: Type[T], *, max_tokens: int = 16000,
                   effort: str = "medium") -> T:
    """Structured generation: the SDK converts the Pydantic model to a JSON schema, the API
    constrains the output to it, and `parsed_output` is a validated instance of `model_cls`."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
```

```
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
        output_format=model_cls,
    ) as stream:
        msg = stream.get_final_message()
    _check(msg)
    if msg.parsed_output is None:  # should not happen with a constrained schema; be defensive
        return model_cls.model_validate_json(_text(msg))
    return msg.parsed_output
```

16.2.3 `main.py`

```
"""
AI Research Discovery Agent — "Your AI Research Scientist"
============================================================
topic → search (arXiv, Semantic Scholar, PubMed, Crossref) → trends → gaps → ideas → proposal

Usage
    python main.py --topic "AI-based drought prediction in wheat"
    python main.py --topic "..." --max-papers 60 --sources arxiv,semantic_scholar
    python main.py --topic "..." --offline          # demo without internet (small built-in sample)

Outputs are written to outputs/<topic-slug>/  (papers.json, trends.json, gaps.json, ideas.json, proposal.md)

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import json
import sys
from pathlib import Path

import typer
from rich.console import Console
from rich.panel import Panel
from rich.table import Table
from slugify import slugify

sys.path.insert(0, str(Path(__file__).parent))
if hasattr(sys.stdout, "reconfigure"):  # Windows consoles default to cp1252; we print → ✓ etc.
    sys.stdout.reconfigure(encoding="utf-8")  # allow `python path/to/main.py`
from agents import (IdeaGeneratorAgent, LiteratureMiningAgent, ProposalWriterAgent,  # noqa: E402
                    ResearchGapAgent)
from tools import OFFLINE_SAMPLE, Paper, dedupe, search_all  # noqa: E402

app = typer.Typer(add_completion=False)
console = Console()
OUT_DIR = Path(__file__).parent / "outputs"

BANNER = """[bold cyan]AI Research Discovery Agent[/] — Your AI Research Scientist
[dim]Codanics · Python ka Chilla: Build AI Agents · https://codanics.com/courses/python-ka-chilla-build-ai-agents/
YouTube @codanics · WhatsApp https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r[/]"""
```

```
def save(path: Path, data) -> None:
    path.parent.mkdir(parents=True, exist_ok=True)
    if isinstance(data, str):
        path.write_text(data, encoding="utf-8")
    else:
        path.write_text(json.dumps(data, indent=2, ensure_ascii=False), encoding="utf-8")
    console.print(f"  [green]saved[/] {path.relative_to(Path(__file__).parent)}")


@app.command()
def run(
    topic: str = typer.Option(..., help="Your research interest in one sentence"),
    max_papers: int = typer.Option(40, help="Approximate number of papers to analyse"),
    sources: str = typer.Option("arxiv,semantic_scholar,pubmed,crossref", help="Comma-separated sources"),
    offline: bool = typer.Option(False, help="Use the built-in demo papers instead of live search"),
    echo: bool = typer.Option(True, help="Stream the proposal to the console while it is written"),
):
    console.print(Panel(BANNER, expand=False))
    out = OUT_DIR / slugify(topic)[:60]
    miner, gapper, ideator, writer = (LiteratureMiningAgent(), ResearchGapAgent(),
                                      IdeaGeneratorAgent(), ProposalWriterAgent())

    # 1) Search ------------------------------------------------------------------
    if offline:
        console.rule("[bold]1. Literature mining (offline demo)")
        papers: list[Paper] = OFFLINE_SAMPLE
    else:
        console.rule("[bold]1. Literature mining")
        plan = miner.plan(topic)
        console.print(f"  queries: {plan.queries}")
        src = [s.strip() for s in sources.split(",") if s.strip()]
        per_source = max(5, max_papers // (len(plan.queries) * len(src)) + 2)
        found: list[Paper] = []
        for q in plan.queries:
            console.print(f"  [cyan]query[/]: {q}")
            found.extend(search_all(q, per_source, src, log=lambda m: console.print(f"    {m}")))
        papers = dedupe(found)[:max_papers]
        if not papers:
            console.print("[yellow]No papers retrieved (network?). Falling back to offline demo sample.[/]")
            papers = OFFLINE_SAMPLE
    save(out / "papers.json", [p.model_dump() for p in papers])

    t = Table(title=f"{len(papers)} papers", show_lines=False)
    t.add_column("id"); t.add_column("year"); t.add_column("title", max_width=80)
    for p in papers[:15]:
        t.add_row(p.id, str(p.year or ""), p.title)
    console.print(t)

    # 2) Trends ------------------------------------------------------------------
    console.rule("[bold]2. Trend analysis")
    trends = miner.analyse(topic, papers)
    console.print(trends.summary)
    save(out / "trends.json", trends.model_dump())

    # 3) Gaps --------------------------------------------------------------------
    console.rule("[bold]3. Research gaps")
    gaps = gapper.run(topic, papers, trends)
    for i, g in enumerate(gaps.gaps):
```

```
        console.print(f"  [bold]{i}.[/] {g.statement}  [dim]{g.evidence_ids}[/]")
    save(out / "gaps.json", gaps.model_dump())

    # 4) Ideas ---------------------------------------------------------------
    console.rule("[bold]4. Research ideas")
    ideas = ideator.run(topic, gaps)
    t = Table(title="Ranked ideas")
    for col in ("#", "title", "novelty", "feasibility", "impact", "suitable for"):
        t.add_column(col)
    for i, idea in enumerate(ideas.ideas, 1):
        t.add_row(str(i), idea.title, str(idea.novelty_score), str(idea.feasibility_score),
                  str(idea.impact_score), idea.suitable_for)
    console.print(t)
    save(out / "ideas.json", ideas.model_dump())

    # 5) Proposal ------------------------------------------------------------
    console.rule("[bold]5. Research proposal (best idea)")
    proposal = writer.run(topic, ideas.ideas[0], gaps, papers, echo=echo)
    save(out / "proposal.md", proposal)
    console.print(Panel(f"Done. All outputs in [bold]{out}[/]", style="green"))


if __name__ == "__main__":
    app()
```

16.2.4 `tools.py`

```
"""
tools.py — real literature-search tools for the Research Discovery Agent.

Every function returns a list[Paper] in ONE common shape, no matter which database it came from.
All four sources are free, public APIs (no key needed; keys only raise rate limits):

    arXiv            https://arxiv.org/help/api
    Semantic Scholar https://api.semanticscholar.org/
    PubMed           https://www.ncbi.nlm.nih.gov/books/NBK25501/  (E-utilities)
    Crossref         https://api.crossref.org/

Golden rule of this project: the LLM may only cite papers that came out of these tools.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import os
import re
import time
import xml.etree.ElementTree as ET
from typing import Iterable

import requests
from dotenv import load_dotenv
from pydantic import BaseModel, Field

load_dotenv()
```

```
USER_AGENT = "codanics-research-agents/1.0 (https://codanics.com)"
TIMEOUT = 30


class Paper(BaseModel):
    """One normalised bibliographic record."""
    id: str = Field(description="Stable identifier, e.g. arxiv:2401.01234, pmid:38012345, doi:10.1000/xyz")
    title: str
    abstract: str = ""
    authors: list[str] = Field(default_factory=list)
    year: int | None = None
    venue: str = ""
    citations: int | None = None
    url: str = ""
    source: str = ""

    def short(self, n_abstract: int = 900) -> str:
        """Compact text block given to the LLM (keeps prompts small and citable)."""
        auth = ", ".join(self.authors[:3]) + (" et al." if len(self.authors) > 3 else "")
        cites = f" | citations: {self.citations}" if self.citations is not None else ""
        abstract = (self.abstract or "")[:n_abstract]
        return f"[{self.id}] {self.title} ({self.year}) — {auth} | {self.venue}{cites}\n{abstract}\n"


# ---------------------------------------------------------------------------- arXiv
def search_arxiv(query: str, max_results: int = 25) -> list[Paper]:
    import arxiv  # pip install arxiv

    client = arxiv.Client(page_size=min(max_results, 100), delay_seconds=3, num_retries=3)
    search = arxiv.Search(query=query, max_results=max_results,
                          sort_by=arxiv.SortCriterion.Relevance)
    papers: list[Paper] = []
    for r in client.results(search):
        papers.append(Paper(
            id=f"arxiv:{r.get_short_id()}",
            title=" ".join(r.title.split()),
            abstract=" ".join(r.summary.split()),
            authors=[a.name for a in r.authors],
            year=r.published.year if r.published else None,
            venue=(r.journal_ref or "arXiv"),
            url=r.entry_id,
            source="arxiv",
        ))
    return papers


# ---------------------------------------------------------------------- Semantic Scholar
def search_semantic_scholar(query: str, max_results: int = 25, year_from: int | None = None) -> list[Paper]:
    headers = {"User-Agent": USER_AGENT}
    key = os.getenv("SEMANTIC_SCHOLAR_API_KEY")
    if key:
        headers["x-api-key"] = key
    params = {
        "query": query,
        "limit": min(max_results, 100),
        "fields": "title,abstract,year,authors,citationCount,externalIds,venue,url",
    }
    if year_from:
```

```
        params["year"] = f"{year_from}-"
    url = "https://api.semanticscholar.org/graph/v1/paper/search"
    for attempt in range(4):  # anonymous access is heavily rate-limited: back off on 429
        resp = requests.get(url, params=params, headers=headers, timeout=TIMEOUT)
        if resp.status_code != 429:
            break
        time.sleep(5 * (attempt + 1))
    resp.raise_for_status()
    papers: list[Paper] = []
    for d in resp.json().get("data", []):
        ext = d.get("externalIds") or {}
        if ext.get("DOI"):
            pid = f"doi:{ext['DOI']}"
        elif ext.get("ArXiv"):
            pid = f"arxiv:{ext['ArXiv']}"
        elif ext.get("PubMed"):
            pid = f"pmid:{ext['PubMed']}"
        else:
            pid = f"s2:{d.get('paperId')}"
        papers.append(Paper(
            id=pid,
            title=d.get("title") or "",
            abstract=d.get("abstract") or "",
            authors=[a.get("name", "") for a in d.get("authors") or []],
            year=d.get("year"),
            venue=d.get("venue") or "",
            citations=d.get("citationCount"),
            url=d.get("url") or "",
            source="semantic_scholar",
        ))
    return papers


# ---------------------------------------------------------------------------- PubMed
def search_pubmed(query: str, max_results: int = 25) -> list[Paper]:
    base = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils"
    common = {"tool": "codanics-research-agents", "email": os.getenv("NCBI_EMAIL", "")}
    if os.getenv("NCBI_API_KEY"):
        common["api_key"] = os.getenv("NCBI_API_KEY")

    ids = requests.get(f"{base}/esearch.fcgi", timeout=TIMEOUT, params={
        **common, "db": "pubmed", "term": query, "retmax": max_results, "retmode": "json",
        "sort": "relevance"}).json().get("esearchresult", {}).get("idlist", [])
    if not ids:
        return []

    xml = requests.get(f"{base}/efetch.fcgi", timeout=TIMEOUT, params={
        **common, "db": "pubmed", "id": ",".join(ids), "retmode": "xml"}).text
    root = ET.fromstring(xml)
    papers: list[Paper] = []
    for art in root.findall(".//PubmedArticle"):
        pmid = art.findtext(".//PMID") or ""
        title = " ".join((art.findtext(".//ArticleTitle") or "").split())
        abstract = " ".join(" ".join(t.itertext()) for t in art.findall(".//AbstractText")).strip()
        authors = []
        for a in art.findall(".//Author"):
            last, fore = a.findtext("LastName"), a.findtext("ForeName")
            if last:
                authors.append(f"{fore} {last}".strip())
```

```
        year = art.findtext(".//PubDate/Year") or art.findtext(".//ArticleDate/Year")
        venue = art.findtext(".//Journal/Title") or ""
        doi = None
        for el in art.findall(".//ArticleId"):
            if el.get("IdType") == "doi":
                doi = el.text
        papers.append(Paper(
            id=f"pmid:{pmid}", title=title, abstract=abstract, authors=authors,
            year=int(year) if year and year.isdigit() else None, venue=venue,
            url=f"https://doi.org/{doi}" if doi else f"https://pubmed.ncbi.nlm.nih.gov/{pmid}/",
            source="pubmed"))
    return papers


# ---------------------------------------------------------------------- Crossref
def search_crossref(query: str, max_results: int = 25) -> list[Paper]:
    resp = requests.get("https://api.crossref.org/works", timeout=TIMEOUT,
                        headers={"User-Agent": USER_AGENT},
                        params={"query": query, "rows": min(max_results, 100),
                                "filter": "type:journal-article",
                                "select": "DOI,title,abstract,author,issued,container-title,is-referenced-by-count,URL"})
    resp.raise_for_status()
    papers: list[Paper] = []
    for it in resp.json().get("message", {}).get("items", []):
        title = " ".join((it.get("title") or [""])[0].split())
        if not title:
            continue
        abstract = re.sub(r"<[^>]+>", " ", it.get("abstract") or "")
        abstract = " ".join(abstract.split())
        authors = [f"{a.get('given', '')} {a.get('family', '')}".strip() for a in it.get("author") or []]
        parts = (it.get("issued") or {}).get("date-parts") or [[None]]
        year = parts[0][0] if parts and parts[0] else None
        papers.append(Paper(
            id=f"doi:{it.get('DOI')}", title=title, abstract=abstract, authors=authors,
            year=year, venue=(it.get("container-title") or [""])[0],
            citations=it.get("is-referenced-by-count"), url=it.get("URL") or "", source="crossref"))
    return papers


# ---------------------------------------------------------------------- utilities
def _norm_title(t: str) -> str:
    return re.sub(r"[^a-z0-9]", "", t.lower())[:80]


def dedupe(papers: Iterable[Paper]) -> list[Paper]:
    """Merge duplicates across databases (same DOI or same normalised title)."""
    seen_ids: set[str] = set()
    seen_titles: set[str] = set()
    out: list[Paper] = []
    for p in papers:
        nt = _norm_title(p.title)
        if p.id in seen_ids or (nt and nt in seen_titles):
            continue
        seen_ids.add(p.id)
        if nt:
            seen_titles.add(nt)
        out.append(p)
    return out
```

```
SOURCES = {
    "arxiv": search_arxiv,
    "semantic_scholar": search_semantic_scholar,
    "pubmed": search_pubmed,
    "crossref": search_crossref,
}


def search_all(query: str, max_per_source: int = 20,
               sources: Iterable[str] = ("arxiv", "semantic_scholar", "pubmed", "crossref"),
               log=print) -> list[Paper]:
    """Query every requested source, tolerate failures, and return a de-duplicated list."""
    found: list[Paper] = []
    for name in sources:
        fn = SOURCES[name]
        try:
            res = fn(query, max_per_source)
            log(f"  {name:<17} {len(res):>3} records")
            found.extend(res)
        except Exception as exc:  # network / rate limit / parse errors must never kill the run
            log(f"  {name:<17} FAILED ({type(exc).__name__}: {exc})")
    papers = dedupe(found)
    log(f"  total after dedupe {len(papers):>3}")
    return papers


# ---------------------------------------------------------------------- offline demo
OFFLINE_SAMPLE: list[Paper] = [
    Paper(id="demo:1", title="Deep learning for drought stress detection in wheat using RGB imagery",
          abstract="A CNN trained on RGB canopy images classifies drought stress in wheat with 92% accuracy under field
↪  conditions. Hyperspectral and genomic data were not used.",
          authors=["A. Khan", "B. Ali"], year=2023, venue="Computers and Electronics in Agriculture", citations=41,
↪  source="demo"),
    Paper(id="demo:2", title="Hyperspectral indices for early drought detection in cereals: a review",
          abstract="Reviews vegetation indices derived from hyperspectral sensors for early drought detection; highlights
↪  the lack of integration with machine learning and genotype information.",
          authors=["C. Zhang"], year=2022, venue="Remote Sensing", citations=88, source="demo"),
    Paper(id="demo:3", title="Genomic prediction of drought tolerance in wheat landraces",
          abstract="GBLUP and random forest models predict drought tolerance from SNP data across 400 landraces;
↪  phenotypic imaging data were unavailable.",
          authors=["D. Rehman", "E. Fatima"], year=2024, venue="Theoretical and Applied Genetics", citations=12,
↪  source="demo"),
    Paper(id="demo:4", title="Explainable machine learning for crop stress phenotyping",
          abstract="Applies SHAP to gradient boosting models of crop stress from multispectral UAV data; calls for
↪  multimodal fusion with genomic markers.",
          authors=["F. Ahmed"], year=2024, venue="Plant Phenomics", citations=9, source="demo"),
    Paper(id="demo:5", title="Transformer models for time-series prediction of soil moisture and crop water stress",
          abstract="A temporal transformer predicts soil moisture and canopy temperature from weather and satellite data,
↪  outperforming LSTM baselines; not validated on wheat genotypes.",
          authors=["G. Li", "H. Wang"], year=2023, venue="Agricultural Water Management", citations=27, source="demo"),
    Paper(id="demo:6", title="Multimodal data fusion in precision agriculture: challenges and opportunities",
          abstract="Surveys fusion of imaging, sensor, weather and omics data; identifies the absence of open benchmark
↪  datasets combining phenomics and genomics for drought.",
          authors=["I. Hussain"], year=2025, venue="Frontiers in Plant Science", citations=5, source="demo"),
]
```

17 Code: Agent 2 — Literature Review

Folder: `02_agents_code/agent_02_literature_review/` — self-contained; run `python agent_02_literature_review/main.py --help`.

17.1 README

17.1.1 `README.md`

````
# Agent 2 — AI Literature Review Agent ("Autonomous Systematic Review Scientist")

A PRISMA-style pipeline: **protocol → multi-database search → screening → extraction → review draft**, with an auditable
↪  trail (CSV/JSON) at every step.

```
question → SearchAgent (protocol: criteria + queries)
         → tools.search_all (arXiv · Semantic Scholar · PubMed · Crossref) → dedupe
         → ScreeningAgent (include/exclude/maybe + reason, batches of 10)
         → prisma.build_counts (PRISMA numbers computed by code)
         → ExtractionAgent (evidence table rows)
         → WritingAgent (review.md — cites only included IDs)
```

## Files

| File | Purpose |
|------|---------|
| `main.py` | CLI (`--question`, `--max-records`, `--sources`, `--offline`, `--from-evidence`) |
| `agents.py` | Search / Screening / Extraction / Writing agents + Pydantic models |
| `prisma.py` | PRISMA counting & flow rendering (pure Python) |
| `tools.py` | Database search + `Paper` model + de-duplication + offline sample |
| `llm.py` | Thin Claude client (`ask`, `ask_structured`) |
| `outputs/<slug>/` | `protocol.json`, `records.json`, `screening.csv`, `evidence.csv`, `prisma.json`, `review.md` |

## Run

```powershell
python agent_02_literature_review\main.py --question "AI methods for plant disease detection" --max-records 80
python agent_02_literature_review\main.py --question "..." --offline
# human-in-the-loop: edit outputs/<slug>/evidence.csv, then
python agent_02_literature_review\main.py --question "..." --from-evidence "agent_02_literature_review\outputs\<slug>"
```

## What this agent teaches

1. **Batch structured outputs** — screening 10 abstracts per call with a guaranteed decision per paper.
2. **Code for numbers, LLM for judgement** — PRISMA counts are computed, never generated.
3. **Human-in-the-loop** — edit the evidence table and regenerate.
4. **Audit trail** — every decision has a reason and is saved.
````

```
## Ideas to extend

- Full-text PDF extraction (`pypdf`), dual screening + Cohen's kappa, meta-analysis (forest plots), BibTeX/Zotero export,
↪  Streamlit UI.

---

**Codanics — Python ka Chilla: Build AI Agents**
Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/ · Website: https://codanics.com · YouTube:
↪  [@codanics](https://www.youtube.com/@codanics)
WhatsApp: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r · Free course:
↪  https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40
```

17.2 Source files

17.2.1 agents.py

```
"""
agents.py — the four sub-agents of the Literature Review Agent (PRISMA pipeline).

    1. SearchAgent      : review protocol (PICO-style question, criteria, database queries)
    2. ScreeningAgent   : include / exclude / maybe + reason, in batches (structured)
    3. ExtractionAgent  : evidence table rows (structured)
    4. WritingAgent     : the literature review draft (free text, cites only included IDs)

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from typing import Literal

from pydantic import BaseModel, Field

from llm import ask, ask_structured
from prisma import PrismaCounts
from tools import Paper

BRAND = ("Codanics — Python ka Chilla: Build AI Agents | "
         "https://codanics.com/courses/python-ka-chilla-build-ai-agents/")

CITE_RULE = ("CITATION RULE: cite ONLY papers given in the input using their exact [id] tags. "
             "Never invent papers, numbers or findings.")


# ============================================================ 1. Search / protocol
class Protocol(BaseModel):
    review_title: str
    research_question: str
    pico_or_scope: str = Field(description="Population/Problem, Intervention/Method, Comparison, Outcome (or scope)")
    inclusion_criteria: list[str]
    exclusion_criteria: list[str]
    queries: list[str] = Field(description="3-6 search strings for the databases")
    year_from: int | None = None
```

```python
class SearchAgent:
    SYSTEM = ("You are a systematic-review methodologist (PRISMA 2020). Write precise protocols with "
              "operational inclusion/exclusion criteria and high-recall search strings.")

    def run(self, question: str) -> Protocol:
        return ask_structured(self.SYSTEM, f"Review question: {question}", Protocol, effort="low")


# ============================================================================ 2. Screening
class Decision(BaseModel):
    paper_id: str
    decision: Literal["include", "exclude", "maybe"]
    reason: str = Field(description="Short reason, e.g. 'Excluded: not about plant disease' or 'Included: CNN on leaf
↪  images'")


class ScreeningBatch(BaseModel):
    decisions: list[Decision]


class ScreeningAgent:
    SYSTEM = ("You are a careful title/abstract screener for a systematic review. Apply the protocol "
              "criteria literally. Use 'maybe' when the abstract is insufficient. Return one decision per paper.")

    def run(self, protocol: Protocol, papers: list[Paper], batch_size: int = 10, log=print) -> list[Decision]:
        decisions: list[Decision] = []
        for i in range(0, len(papers), batch_size):
            batch = papers[i:i + batch_size]
            corpus = "\n".join(p.short(1200) for p in batch)
            res = ask_structured(
                self.SYSTEM,
                f"Protocol:\n{protocol.model_dump_json(indent=1)}\n\nPapers to screen:\n{corpus}\n\n"
                f"Return exactly {len(batch)} decisions with the given paper ids.",
                ScreeningBatch, effort="low")
            got = {d.paper_id: d for d in res.decisions}
            for p in batch:  # guarantee one decision per paper even if the model skipped one
                decisions.append(got.get(p.id, Decision(paper_id=p.id, decision="maybe",
                                                        reason="No decision returned; needs human check")))
            log(f"  screened {min(i + batch_size, len(papers))}/{len(papers)}")
        return decisions


# ============================================================================ 3. Extraction
class EvidenceRow(BaseModel):
    paper_id: str
    study: str = Field(description="First author + year, e.g. 'Smith 2025'")
    objective: str
    dataset_or_sample: str
    method: str
    key_result: str = Field(description="Main quantitative/qualitative result, e.g. '94% accuracy'")
    limitations: str
    theme: str = Field(description="Short theme label for grouping, e.g. 'CNN on leaf images'")


class EvidenceTable(BaseModel):
    rows: list[EvidenceRow]
```

```
class ExtractionAgent:
    SYSTEM = ("You extract structured evidence from abstracts for a systematic review. If a field is not "
              "reported, write 'not reported'. " + CITE_RULE)

    def run(self, protocol: Protocol, papers: list[Paper], batch_size: int = 8, log=print) -> list[EvidenceRow]:
        rows: list[EvidenceRow] = []
        for i in range(0, len(papers), batch_size):
            batch = papers[i:i + batch_size]
            corpus = "\n".join(p.short(1500) for p in batch)
            res = ask_structured(
                self.SYSTEM,
                f"Review question: {protocol.research_question}\n\nPapers:\n{corpus}\n\n"
                f"Return one row per paper ({len(batch)} rows).",
                EvidenceTable, effort="medium")
            rows.extend(res.rows)
            log(f"  extracted {min(i + batch_size, len(papers))}/{len(papers)}")
        return rows


# ============================================================================ 4. Writing
class WritingAgent:
    SYSTEM = ("You are an academic writer producing a systematic literature review in Markdown. "
              + CITE_RULE + " Structure: Title; Abstract; 1 Introduction; 2 Methods (PRISMA: databases, "
              "queries, criteria, screening, extraction); 3 Results (3.1 Study characteristics, "
              "3.2-3.x Thematic synthesis with an evidence table); 4 Research Gaps; 5 Discussion; "
              "6 Conclusion; References (only cited ids, one per line with title). "
              "Be specific and quantitative; do not pad.")

    def run(self, protocol: Protocol, prisma: PrismaCounts, rows: list[EvidenceRow],
            included: list[Paper], echo: bool = False) -> str:
        table = "\n".join(r.model_dump_json() for r in rows)
        refs = "\n".join(f"[{p.id}] {p.title} ({p.year}) {p.venue}" for p in included)
        text = ask(
            self.SYSTEM,
            f"Protocol:\n{protocol.model_dump_json(indent=1)}\n\nPRISMA counts (use these numbers verbatim):\n"
            f"{prisma.model_dump_json(indent=1)}\n\nEvidence table rows:\n{table}\n\n"
            f"Included papers (for citation):\n{refs}\n\nWrite the full review.",
            max_tokens=32000, effort="high", echo=echo)
        return text + f"\n\n---\n*Generated with the AI Literature Review Agent — {BRAND}*\n"
```

17.2.2 `llm.py`

```
"""
llm.py — a thin wrapper around the Anthropic Claude API used by this agent.

Two functions are all you need:
    ask(system, user)                  -> free text (streamed)
    ask_structured(system, user, Model) -> a validated Pydantic object (JSON structured output)

Why a wrapper?  Every agent in this project talks to the model the same way.  Keeping the
API details in ONE file per agent means you can swap the model or provider in one place.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
```

```python
from __future__ import annotations

import os
from typing import Type, TypeVar

import anthropic
from dotenv import load_dotenv
from pydantic import BaseModel

load_dotenv()  # reads ANTHROPIC_API_KEY / CLAUDE_MODEL from a .env file if present

MODEL = os.getenv("CLAUDE_MODEL", "claude-opus-5")
T = TypeVar("T", bound=BaseModel)

_client: anthropic.Anthropic | None = None


def client() -> anthropic.Anthropic:
    """Lazily create one shared client (credentials come from the environment)."""
    global _client
    if _client is None:
        _client = anthropic.Anthropic()
    return _client


def _check(msg: anthropic.types.Message) -> None:
    if msg.stop_reason == "refusal":
        detail = getattr(msg, "stop_details", None)
        raise RuntimeError(f"The model refused this request: {detail}")
    if msg.stop_reason == "max_tokens":
        raise RuntimeError("Output was cut off (max_tokens). Increase max_tokens and retry.")


def _text(msg: anthropic.types.Message) -> str:
    return "".join(b.text for b in msg.content if b.type == "text")


def ask(system: str, user: str, *, max_tokens: int = 16000, effort: str = "medium",
        echo: bool = False) -> str:
    """Free-text generation. Streams so long outputs never hit HTTP timeouts."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
    ) as stream:
        if echo:
            for chunk in stream.text_stream:
                print(chunk, end="", flush=True)
            print()
        msg = stream.get_final_message()
    _check(msg)
    return _text(msg)


def ask_structured(system: str, user: str, model_cls: Type[T], *, max_tokens: int = 16000,
                   effort: str = "medium") -> T:
```

```
    """Structured generation: the SDK converts the Pydantic model to a JSON schema, the API
    constrains the output to it, and `parsed_output` is a validated instance of `model_cls`."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
        output_format=model_cls,
    ) as stream:
        msg = stream.get_final_message()
    _check(msg)
    if msg.parsed_output is None:  # should not happen with a constrained schema; be defensive
        return model_cls.model_validate_json(_text(msg))
    return msg.parsed_output
```

17.2.3 `main.py`

```
"""
AI Literature Review Agent — "Autonomous Systematic Review Scientist"
=====================================================================
question → protocol → search (4 databases) → screen → extract → PRISMA counts → review.md

Usage
    python main.py --question "AI methods for plant disease detection"
    python main.py --question "..." --max-records 80 --sources arxiv,pubmed
    python main.py --question "..." --offline                  # demo sample, no internet for search
    python main.py --question "..." --from-evidence outputs/<slug>   # re-write after editing evidence.csv

Outputs: outputs/<slug>/  protocol.json, records.json, screening.csv, evidence.csv, prisma.json, review.md

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import csv
import json
import sys
from pathlib import Path

import typer
from rich.console import Console
from rich.panel import Panel
from rich.table import Table
from slugify import slugify

sys.path.insert(0, str(Path(__file__).parent))
if hasattr(sys.stdout, "reconfigure"):  # Windows consoles default to cp1252; we print → ✓ etc.
    sys.stdout.reconfigure(encoding="utf-8")
from agents import (EvidenceRow, ExtractionAgent, Protocol, ScreeningAgent,  # noqa: E402
                    SearchAgent, WritingAgent)
from prisma import PrismaCounts, build_counts  # noqa: E402
from tools import OFFLINE_SAMPLE, Paper, dedupe, search_all  # noqa: E402
```

```
app = typer.Typer(add_completion=False)
console = Console()
OUT_DIR = Path(__file__).parent / "outputs"

BANNER = """[bold cyan]AI Literature Review Agent[/] — Autonomous Systematic Review Scientist
[dim]Codanics · Python ka Chilla: Build AI Agents · https://codanics.com/courses/python-ka-chilla-build-ai-agents/
YouTube @codanics · WhatsApp https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r[/]"""


def write_csv(path: Path, rows: list[dict]) -> None:
    path.parent.mkdir(parents=True, exist_ok=True)
    if not rows:
        path.write_text("", encoding="utf-8")
        return
    with path.open("w", newline="", encoding="utf-8") as f:
        w = csv.DictWriter(f, fieldnames=list(rows[0].keys()))
        w.writeheader()
        w.writerows(rows)
    console.print(f"  [green]saved[/] {path.name}")


def write_json(path: Path, data) -> None:
    path.parent.mkdir(parents=True, exist_ok=True)
    path.write_text(json.dumps(data, indent=2, ensure_ascii=False), encoding="utf-8")
    console.print(f"  [green]saved[/] {path.name}")


@app.command()
def run(
    question: str = typer.Option(..., help="The review question / topic"),
    max_records: int = typer.Option(60, help="Max records to screen"),
    sources: str = typer.Option("arxiv,semantic_scholar,pubmed,crossref"),
    offline: bool = typer.Option(False, help="Use built-in demo records"),
    from_evidence: Path = typer.Option(None, help="Existing output folder: skip to writing using its evidence.csv"),
    echo: bool = typer.Option(True, help="Stream the review to the console"),
):
    console.print(Panel(BANNER, expand=False))
    out = from_evidence or (OUT_DIR / slugify(question)[:60])

    # --- resume mode: human edited evidence.csv, just re-write the review -------------
    if from_evidence:
        console.rule("[bold]Re-writing review from edited evidence")
        protocol = Protocol.model_validate_json((out / "protocol.json").read_text(encoding="utf-8"))
        prisma = PrismaCounts.model_validate_json((out / "prisma.json").read_text(encoding="utf-8"))
        rows = [EvidenceRow(**r) for r in csv.DictReader((out / "evidence.csv").open(encoding="utf-8"))]
        records = [Paper(**p) for p in json.loads((out / "records.json").read_text(encoding="utf-8"))]
        included = [p for p in records if p.id in {r.paper_id for r in rows}]
        review = WritingAgent().run(protocol, prisma, rows, included, echo=echo)
        (out / "review.md").write_text(review, encoding="utf-8")
        console.print(Panel(f"Done → {out / 'review.md'}", style="green"))
        return

    # 1) Protocol ------------------------------------------------------------------
    console.rule("[bold]1. Search strategy / protocol")
    protocol = SearchAgent().run(question)
    console.print(f"  [bold]{protocol.review_title}[/]\n  Q: {protocol.research_question}")
    console.print(f"  include: {protocol.inclusion_criteria}\n  exclude: {protocol.exclusion_criteria}")
    write_json(out / "protocol.json", protocol.model_dump())
```

```
    # 2) Search ---------------------------------------------------------------
    console.rule("[bold]2. Database search")
    if offline:
        raw = list(OFFLINE_SAMPLE)
    else:
        src = [s.strip() for s in sources.split(",") if s.strip()]
        per = max(5, max_records // (len(protocol.queries) * len(src)) + 2)
        raw = []
        for q in protocol.queries:
            console.print(f"  [cyan]query[/]: {q}")
            raw.extend(search_all(q, per, src, log=lambda m: console.print(f"    {m}")))
        if not raw:
            console.print("[yellow]Nothing retrieved — using offline sample.[/]")
            raw = list(OFFLINE_SAMPLE)
    records = dedupe(raw)[:max_records]
    write_json(out / "records.json", [p.model_dump() for p in records])
    console.print(f"  identified {len(raw)} → after de-duplication {len(records)}")

    # 3) Screening ------------------------------------------------------------
    console.rule("[bold]3. Title/abstract screening")
    decisions = ScreeningAgent().run(protocol, records, log=lambda m: console.print(f"  {m}"))
    write_csv(out / "screening.csv", [d.model_dump() for d in decisions])
    prisma = build_counts(raw, records, decisions)
    write_json(out / "prisma.json", prisma.model_dump())
    console.print(prisma.flow_markdown())

    included_ids = {d.paper_id for d in decisions if d.decision == "include"}
    included = [p for p in records if p.id in included_ids]
    if not included:
        console.print("[red]No papers included. Loosen criteria or increase --max-records.[/]")
        raise typer.Exit(1)

    # 4) Extraction -----------------------------------------------------------
    console.rule("[bold]4. Knowledge extraction")
    rows = ExtractionAgent().run(protocol, included, log=lambda m: console.print(f"  {m}"))
    write_csv(out / "evidence.csv", [r.model_dump() for r in rows])
    t = Table(title="Evidence table (first rows)")
    for c in ("study", "dataset_or_sample", "method", "key_result"):
        t.add_column(c, max_width=40)
    for r in rows[:10]:
        t.add_row(r.study, r.dataset_or_sample, r.method, r.key_result)
    console.print(t)

    # 5) Writing --------------------------------------------------------------
    console.rule("[bold]5. Writing the review")
    review = WritingAgent().run(protocol, prisma, rows, included, echo=echo)
    (out / "review.md").write_text(review, encoding="utf-8")
    console.print(Panel(f"Done. Outputs in [bold]{out}[/]\nEdit evidence.csv and re-run with "
                        f"--from-evidence \"{out}\" to regenerate the review.", style="green"))


if __name__ == "__main__":
    app()
```

17.2.4 `prisma.py`

```
"""
prisma.py — PRISMA flow bookkeeping, done by CODE not by the LLM.

PRISMA (Preferred Reporting Items for Systematic reviews and Meta-Analyses) asks reviewers to
report how many records were identified, de-duplicated, screened, excluded and included.
We count these deterministically so the numbers in the review are always correct.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from collections import Counter

from pydantic import BaseModel, Field


class PrismaCounts(BaseModel):
    identified_per_source: dict[str, int] = Field(default_factory=dict)
    identified_total: int = 0
    duplicates_removed: int = 0
    screened: int = 0
    excluded: int = 0
    maybe: int = 0
    included: int = 0
    exclusion_reasons: dict[str, int] = Field(default_factory=dict)

    def flow_markdown(self) -> str:
        src = ", ".join(f"{k}: {v}" for k, v in self.identified_per_source.items()) or "n/a"
        reasons = "\n".join(f"  - {r}: {n}" for r, n in self.exclusion_reasons.items()) or "  - (none)"
        return f"""```
Identification   Records identified from databases: {self.identified_total}
                 ({src})
                              |
                 Duplicates removed: {self.duplicates_removed}
                              ▼
Screening        Records screened (title/abstract): {self.screened}
                              |
                 Records excluded: {self.excluded}   |  Uncertain / needs human check: {self.maybe}
                              ▼
Included         Studies included in the review: {self.included}
```
Exclusion reasons:
{reasons}
"""

def build_counts(raw: list, deduped: list, decisions: list) -> PrismaCounts:
    """`decisions` is a list of objects with `.decision` in {include, exclude, maybe} and `.reason`."""
    per_source = Counter(getattr(p, "source", "unknown") for p in raw)
    dec = Counter(d.decision for d in decisions)
    reasons = Counter(d.reason.split(":")[0].strip()[:60] for d in decisions if d.decision == "exclude")
    return PrismaCounts(
        identified_per_source=dict(per_source),
        identified_total=len(raw),
        duplicates_removed=len(raw) - len(deduped),
```
```

```
        screened=len(decisions),
        excluded=dec.get("exclude", 0),
        maybe=dec.get("maybe", 0),
        included=dec.get("include", 0),
        exclusion_reasons=dict(reasons.most_common(10)),
    )
```

17.2.5 `tools.py`

```
"""
tools.py — real literature-search tools for the Literature Review Agent.

Every function returns a list[Paper] in ONE common shape, no matter which database it came from.
All four sources are free, public APIs (no key needed; keys only raise rate limits):

    arXiv            https://arxiv.org/help/api
    Semantic Scholar https://api.semanticscholar.org/
    PubMed           https://www.ncbi.nlm.nih.gov/books/NBK25501/  (E-utilities)
    Crossref         https://api.crossref.org/

Golden rule of this project: the LLM may only cite papers that came out of these tools.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import os
import re
import time
import xml.etree.ElementTree as ET
from typing import Iterable

import requests
from dotenv import load_dotenv
from pydantic import BaseModel, Field

load_dotenv()

USER_AGENT = "codanics-research-agents/1.0 (https://codanics.com)"
TIMEOUT = 30


class Paper(BaseModel):
    """One normalised bibliographic record."""
    id: str = Field(description="Stable identifier, e.g. arxiv:2401.01234, pmid:38012345, doi:10.1000/xyz")
    title: str
    abstract: str = ""
    authors: list[str] = Field(default_factory=list)
    year: int | None = None
    venue: str = ""
    citations: int | None = None
    url: str = ""
    source: str = ""

    def short(self, n_abstract: int = 900) -> str:
```

```
        """Compact text block given to the LLM (keeps prompts small and citable)."""
        auth = ", ".join(self.authors[:3]) + (" et al." if len(self.authors) > 3 else "")
        cites = f" | citations: {self.citations}" if self.citations is not None else ""
        abstract = (self.abstract or "")[:n_abstract]
        return f"[{self.id}] {self.title} ({self.year}) — {auth} | {self.venue}{cites}\n{abstract}\n"


# ---------------------------------------------------------------------------- arXiv
def search_arxiv(query: str, max_results: int = 25) -> list[Paper]:
    import arxiv  # pip install arxiv

    client = arxiv.Client(page_size=min(max_results, 100), delay_seconds=3, num_retries=3)
    search = arxiv.Search(query=query, max_results=max_results,
                          sort_by=arxiv.SortCriterion.Relevance)
    papers: list[Paper] = []
    for r in client.results(search):
        papers.append(Paper(
            id=f"arxiv:{r.get_short_id()}",
            title=" ".join(r.title.split()),
            abstract=" ".join(r.summary.split()),
            authors=[a.name for a in r.authors],
            year=r.published.year if r.published else None,
            venue=(r.journal_ref or "arXiv"),
            url=r.entry_id,
            source="arxiv",
        ))
    return papers


# ---------------------------------------------------------------------- Semantic Scholar
def search_semantic_scholar(query: str, max_results: int = 25, year_from: int | None = None) -> list[Paper]:
    headers = {"User-Agent": USER_AGENT}
    key = os.getenv("SEMANTIC_SCHOLAR_API_KEY")
    if key:
        headers["x-api-key"] = key
    params = {
        "query": query,
        "limit": min(max_results, 100),
        "fields": "title,abstract,year,authors,citationCount,externalIds,venue,url",
    }
    if year_from:
        params["year"] = f"{year_from}-"
    url = "https://api.semanticscholar.org/graph/v1/paper/search"
    for attempt in range(4):  # anonymous access is heavily rate-limited: back off on 429
        resp = requests.get(url, params=params, headers=headers, timeout=TIMEOUT)
        if resp.status_code != 429:
            break
        time.sleep(5 * (attempt + 1))
    resp.raise_for_status()
    papers: list[Paper] = []
    for d in resp.json().get("data", []):
        ext = d.get("externalIds") or {}
        if ext.get("DOI"):
            pid = f"doi:{ext['DOI']}"
        elif ext.get("ArXiv"):
            pid = f"arxiv:{ext['ArXiv']}"
        elif ext.get("PubMed"):
            pid = f"pmid:{ext['PubMed']}"
        else:
```

```
            pid = f"s2:{d.get('paperId')}"
        papers.append(Paper(
            id=pid,
            title=d.get("title") or "",
            abstract=d.get("abstract") or "",
            authors=[a.get("name", "") for a in d.get("authors") or []],
            year=d.get("year"),
            venue=d.get("venue") or "",
            citations=d.get("citationCount"),
            url=d.get("url") or "",
            source="semantic_scholar",
        ))
    return papers


# ---------------------------------------------------------------------- PubMed
def search_pubmed(query: str, max_results: int = 25) -> list[Paper]:
    base = "https://eutils.ncbi.nlm.nih.gov/entrez/eutils"
    common = {"tool": "codanics-research-agents", "email": os.getenv("NCBI_EMAIL", "")}
    if os.getenv("NCBI_API_KEY"):
        common["api_key"] = os.getenv("NCBI_API_KEY")

    ids = requests.get(f"{base}/esearch.fcgi", timeout=TIMEOUT, params={
        **common, "db": "pubmed", "term": query, "retmax": max_results, "retmode": "json",
        "sort": "relevance"}).json().get("esearchresult", {}).get("idlist", [])
    if not ids:
        return []

    xml = requests.get(f"{base}/efetch.fcgi", timeout=TIMEOUT, params={
        **common, "db": "pubmed", "id": ",".join(ids), "retmode": "xml"}).text
    root = ET.fromstring(xml)
    papers: list[Paper] = []
    for art in root.findall(".//PubmedArticle"):
        pmid = art.findtext(".//PMID") or ""
        title = " ".join((art.findtext(".//ArticleTitle") or "").split())
        abstract = " ".join(" ".join(t.itertext()) for t in art.findall(".//AbstractText")).strip()
        authors = []
        for a in art.findall(".//Author"):
            last, fore = a.findtext("LastName"), a.findtext("ForeName")
            if last:
                authors.append(f"{fore} {last}".strip())
        year = art.findtext(".//PubDate/Year") or art.findtext(".//ArticleDate/Year")
        venue = art.findtext(".//Journal/Title") or ""
        doi = None
        for el in art.findall(".//ArticleId"):
            if el.get("IdType") == "doi":
                doi = el.text
        papers.append(Paper(
            id=f"pmid:{pmid}", title=title, abstract=abstract, authors=authors,
            year=int(year) if year and year.isdigit() else None, venue=venue,
            url=f"https://doi.org/{doi}" if doi else f"https://pubmed.ncbi.nlm.nih.gov/{pmid}/",
            source="pubmed"))
    return papers


# ---------------------------------------------------------------------- Crossref
def search_crossref(query: str, max_results: int = 25) -> list[Paper]:
    resp = requests.get("https://api.crossref.org/works", timeout=TIMEOUT,
                        headers={"User-Agent": USER_AGENT},
```

```
                        params={"query": query, "rows": min(max_results, 100),
                                "filter": "type:journal-article",
                                "select": "DOI,title,abstract,author,issued,container-title,is-referenced-by-count,URL"})
    resp.raise_for_status()
    papers: list[Paper] = []
    for it in resp.json().get("message", {}).get("items", []):
        title = " ".join((it.get("title") or [""])[0].split())
        if not title:
            continue
        abstract = re.sub(r"<[^>]+>", " ", it.get("abstract") or "")
        abstract = " ".join(abstract.split())
        authors = [f"{a.get('given', '')} {a.get('family', '')}".strip() for a in it.get("author") or []]
        parts = (it.get("issued") or {}).get("date-parts") or [[None]]
        year = parts[0][0] if parts and parts[0] else None
        papers.append(Paper(
            id=f"doi:{it.get('DOI')}", title=title, abstract=abstract, authors=authors,
            year=year, venue=(it.get("container-title") or [""])[0],
            citations=it.get("is-referenced-by-count"), url=it.get("URL") or "", source="crossref"))
    return papers


# ---------------------------------------------------------------------- utilities
def _norm_title(t: str) -> str:
    return re.sub(r"[^a-z0-9]", "", t.lower())[:80]


def dedupe(papers: Iterable[Paper]) -> list[Paper]:
    """Merge duplicates across databases (same DOI or same normalised title)."""
    seen_ids: set[str] = set()
    seen_titles: set[str] = set()
    out: list[Paper] = []
    for p in papers:
        nt = _norm_title(p.title)
        if p.id in seen_ids or (nt and nt in seen_titles):
            continue
        seen_ids.add(p.id)
        if nt:
            seen_titles.add(nt)
        out.append(p)
    return out


SOURCES = {
    "arxiv": search_arxiv,
    "semantic_scholar": search_semantic_scholar,
    "pubmed": search_pubmed,
    "crossref": search_crossref,
}


def search_all(query: str, max_per_source: int = 20,
               sources: Iterable[str] = ("arxiv", "semantic_scholar", "pubmed", "crossref"),
               log=print) -> list[Paper]:
    """Query every requested source, tolerate failures, and return a de-duplicated list."""
    found: list[Paper] = []
    for name in sources:
        fn = SOURCES[name]
        try:
            res = fn(query, max_per_source)
```

```
            log(f"  {name:<17} {len(res):>3} records")
            found.extend(res)
        except Exception as exc:  # network / rate limit / parse errors must never kill the run
            log(f"  {name:<17} FAILED ({type(exc).__name__}: {exc})")
    papers = dedupe(found)
    log(f"  total after dedupe {len(papers):>3}")
    return papers


# ---------------------------------------------------------------------- offline demo
OFFLINE_SAMPLE: list[Paper] = [
    Paper(id="demo:1", title="Deep learning for drought stress detection in wheat using RGB imagery",
          abstract="A CNN trained on RGB canopy images classifies drought stress in wheat with 92% accuracy under field
↪  conditions. Hyperspectral and genomic data were not used.",
          authors=["A. Khan", "B. Ali"], year=2023, venue="Computers and Electronics in Agriculture", citations=41,
↪  source="demo"),
    Paper(id="demo:2", title="Hyperspectral indices for early drought detection in cereals: a review",
          abstract="Reviews vegetation indices derived from hyperspectral sensors for early drought detection; highlights
↪  the lack of integration with machine learning and genotype information.",
          authors=["C. Zhang"], year=2022, venue="Remote Sensing", citations=88, source="demo"),
    Paper(id="demo:3", title="Genomic prediction of drought tolerance in wheat landraces",
          abstract="GBLUP and random forest models predict drought tolerance from SNP data across 400 landraces;
↪  phenotypic imaging data were unavailable.",
          authors=["D. Rehman", "E. Fatima"], year=2024, venue="Theoretical and Applied Genetics", citations=12,
↪  source="demo"),
    Paper(id="demo:4", title="Explainable machine learning for crop stress phenotyping",
          abstract="Applies SHAP to gradient boosting models of crop stress from multispectral UAV data; calls for
↪  multimodal fusion with genomic markers.",
          authors=["F. Ahmed"], year=2024, venue="Plant Phenomics", citations=9, source="demo"),
    Paper(id="demo:5", title="Transformer models for time-series prediction of soil moisture and crop water stress",
          abstract="A temporal transformer predicts soil moisture and canopy temperature from weather and satellite data,
↪  outperforming LSTM baselines; not validated on wheat genotypes.",
          authors=["G. Li", "H. Wang"], year=2023, venue="Agricultural Water Management", citations=27, source="demo"),
    Paper(id="demo:6", title="Multimodal data fusion in precision agriculture: challenges and opportunities",
          abstract="Surveys fusion of imaging, sensor, weather and omics data; identifies the absence of open benchmark
↪  datasets combining phenomics and genomics for drought.",
          authors=["I. Hussain"], year=2025, venue="Frontiers in Plant Science", citations=5, source="demo"),
]
```

18 Code: Agent 3 — Research Experiment

Folder: `02_agents_code/agent_03_research_experiment/` — self-contained; run `python agent_03_research_experiment/main.py --help`.

18.1 README

18.1.1 `README.md`

```
# Agent 3 — AI Research Experiment Agent ("Virtual Research Assistant")

**Code does the maths, the LLM does the reasoning.** The agent plans the experiment, runs a deterministic analysis
↪  pipeline (QC → normalisation → Welch t-test + BH-FDR → PCA/volcano → ML with cross-validation), asks a "statistician"
↪  agent to review the numbers, and writes the Methods & Results plus a reproducible notebook.

```
description (+ CSV or --demo)
   ─► ExperimentPlanner (LLM, structured)      → design.json
   ─► analysis.py (pandas/scipy/statsmodels/sklearn) → summary.json, differential.csv, pca.png, volcano.png
   ─► StatisticsAgent (LLM, structured)        → stats_review.json
   ─► ReportWriter (LLM)                        → methods_results.md
   ─► notebook.py (nbformat)                    → analysis_notebook.ipynb
```

Files

File	Purpose
`main.py`	CLI (`--description`, `--data`, `--group-col`, `--demo`, `--no-llm`)
`data.py`	Load a CSV (samples as rows) or generate synthetic RNA-seq-like counts
`analysis.py`	QC, normalisation, differential test, figures, ML baseline, summary — **no LLM**
`agents.py`	Planner, Statistics reviewer, Report writer
`notebook.py`	Generates a runnable Jupyter notebook
`llm.py`	Thin Claude client

Run

```powershell
python agent_03_research_experiment\main.py --description "RNA-seq of heat-stressed rice, control vs heat, n=3" --demo
python agent_03_research_experiment\main.py --description "..." --data mydata.csv --group-col treatment
python agent_03_research_experiment\main.py --description "..." --demo --no-llm     # no API key needed
```

CSV layout: `sample_id, <group_col>, feature_1, feature_2, ...` — two groups for the differential test.

What this agent teaches

1. **Separation of concerns** — LLM for planning/critique/writing; Python for numbers.
2. **Structured critique** — a reviewer agent that returns concerns with severity and recommendations.
```

```
3. **Reproducibility as a feature** — notebook + CSVs + PNGs alongside the prose.
4. **Privacy-friendly design** — only summary statistics are sent to the model, never raw data.

## Ideas to extend

- Domain plugins: DESeq2/pydeseq2 for RNA-seq, ANOVA/mixed models for field trials, survival analysis for clinical data.
- Sandboxed code execution so the agent can write custom analysis code.
- Power/sample-size calculator at design time.

---

**Codanics — Python ka Chilla: Build AI Agents**
Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/ · Website: https://codanics.com · YouTube:
↪ [@codanics](https://www.youtube.com/@codanics)
WhatsApp: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r · Free course:
↪ https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40
```

18.2 Source files

18.2.1 `agents.py`

```
"""
agents.py — the LLM-powered sub-agents of the Research Experiment Agent.

    ExperimentPlanner : description → ExperimentDesign (structured)
    StatisticsAgent   : analysis summary → StatsReview   (structured critique)
    ReportWriter      : design + numbers + critique → Methods & Results (Markdown)

The heavy lifting (QC, tests, ML, figures) lives in analysis.py and never touches the LLM.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import json

from pydantic import BaseModel, Field

from llm import ask, ask_structured

BRAND = ("Codanics — Python ka Chilla: Build AI Agents | "
         "https://codanics.com/courses/python-ka-chilla-build-ai-agents/")


# ---------------------------------------------------------------------- 1. Planner
class ExperimentDesign(BaseModel):
    title: str
    hypothesis: str
    design_type: str = Field(description="e.g. two-group comparison, factorial, time-series, RCT")
    groups: list[str]
    replicates_per_group: int
    controls: list[str]
    covariates_or_blocking: list[str]
    sample_size_note: str = Field(description="Comment on power / adequacy of replicates")
```

```
    analysis_plan: list[str] = Field(description="Ordered steps: QC, normalisation, tests, ML, validation")
    statistical_tests: list[str]
    risks: list[str]


class ExperimentPlanner:
    SYSTEM = ("You are a biostatistician and experimental-design expert. Produce a concrete, minimal "
              "design that a lab can execute and a reviewer will accept.")

    def run(self, description: str, data_hint: str = "") -> ExperimentDesign:
        return ask_structured(self.SYSTEM,
                              f"Experiment description: {description}\n{data_hint}\nDesign the experiment and analysis
                              ↪  plan.",
                              ExperimentDesign, effort="medium")


# ------------------------------------------------------------------ 2. Statistics
class Concern(BaseModel):
    issue: str
    severity: str = Field(description="low | medium | high")
    recommendation: str


class StatsReview(BaseModel):
    assumptions_checked: list[str]
    sample_size_adequate: bool
    multiple_testing_handled: bool
    leakage_or_bias_risk: str
    concerns: list[Concern]
    interpretation: str = Field(description="Plain-language interpretation of the key numbers")
    overall_verdict: str = Field(description="e.g. 'Preliminary evidence; needs biological validation'")


class StatisticsAgent:
    SYSTEM = ("You are a rigorous reviewing statistician. You receive ONLY computed numbers (never raw data). "
              "Check assumptions, sample size, multiple testing, leakage and over-interpretation. Be concrete.")

    def run(self, design: ExperimentDesign, summary: dict) -> StatsReview:
        return ask_structured(self.SYSTEM,
                              f"Design:\n{design.model_dump_json(indent=1)}\n\nAnalysis summary:\n"
                              f"{json.dumps(summary, indent=1)}\n\nReview it.",
                              StatsReview, effort="high")


# ------------------------------------------------------------------ 3. Report
class ReportWriter:
    SYSTEM = ("You write the Methods and Results sections of a scientific paper in Markdown, using ONLY the "
              "numbers provided (never invent values). Include: Methods (design, samples, normalisation, "
              "statistical tests, ML & validation, software), Results (with a compact table of top features "
              "and references to Figure 1 (PCA) and Figure 2 (volcano)), Limitations, and a short "
              "'Statistical review' box summarising the reviewer's concerns.")

    def run(self, description: str, design: ExperimentDesign, summary: dict, review: StatsReview,
            echo: bool = False) -> str:
        text = ask(self.SYSTEM,
                   f"Study: {description}\n\nDesign:\n{design.model_dump_json(indent=1)}\n\n"
                   f"Computed results:\n{json.dumps(summary, indent=1)}\n\nStatistical review:\n"
                   f"{review.model_dump_json(indent=1)}",
                   effort="high", echo=echo)
```

```
        return text + f"\n\n---\n*Generated with the AI Research Experiment Agent — {BRAND}*\n"
```

18.2.2 `analysis.py`

```
"""
analysis.py — deterministic analysis engine (QC → normalisation → differential test → ML).

Nothing in this file calls the LLM.  Numbers, p-values and figures come from pandas / scipy /
statsmodels / scikit-learn so that results are reproducible and defensible.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from pathlib import Path

import matplotlib
matplotlib.use("Agg")  # headless
import matplotlib.pyplot as plt  # noqa: E402
import numpy as np  # noqa: E402
import pandas as pd  # noqa: E402
from scipy import stats  # noqa: E402
from sklearn.decomposition import PCA  # noqa: E402
from sklearn.ensemble import RandomForestClassifier  # noqa: E402
from sklearn.linear_model import LogisticRegression  # noqa: E402
from sklearn.model_selection import LeaveOneOut, StratifiedKFold, cross_val_score  # noqa: E402
from sklearn.pipeline import make_pipeline  # noqa: E402
from sklearn.preprocessing import StandardScaler  # noqa: E402
from statsmodels.stats.multitest import multipletests  # noqa: E402

META_COLS = {"sample_id"}


def feature_matrix(df: pd.DataFrame, group_col: str) -> pd.DataFrame:
    cols = [c for c in df.columns if c not in META_COLS | {group_col}]
    return df[cols].apply(pd.to_numeric, errors="coerce")


# ---------------------------------------------------------------------------- QC
def quality_control(df: pd.DataFrame, group_col: str) -> dict:
    X = feature_matrix(df, group_col)
    missing = int(X.isna().sum().sum())
    zero_var = X.columns[X.var(axis=0).fillna(0) == 0].tolist()
    per_sample_total = X.sum(axis=1)
    return {
        "n_samples": int(len(df)),
        "n_features": int(X.shape[1]),
        "groups": df[group_col].value_counts().to_dict(),
        "missing_values": missing,
        "zero_variance_features": len(zero_var),
        "library_size_min": float(per_sample_total.min()),
        "library_size_max": float(per_sample_total.max()),
        "library_size_cv": float(per_sample_total.std() / per_sample_total.mean()) if per_sample_total.mean() else 0.0,
        "warnings": ([f"{missing} missing values (filled with column median)"] if missing else [])
                    + ([f"{len(zero_var)} zero-variance features removed"] if zero_var else [])
```

```
                    + (["fewer than 3 replicates in a group — statistics are unreliable"]
                       if df[group_col].value_counts().min() < 3 else []),
    }


def normalise(df: pd.DataFrame, group_col: str) -> pd.DataFrame:
    """Median-fill, drop zero-variance, then log2(CPM+1) if data look like counts, else z-score."""
    X = feature_matrix(df, group_col)
    X = X.fillna(X.median())
    X = X.loc[:, X.var(axis=0) > 0]
    looks_like_counts = (X.values >= 0).all() and (X.values.max() > 50)
    if looks_like_counts:
        cpm = X.div(X.sum(axis=1), axis=0) * 1e6
        X = np.log2(cpm + 1)
    else:
        X = (X - X.mean()) / X.std(ddof=0).replace(0, 1)
    out = pd.concat([df[["sample_id", group_col]].reset_index(drop=True), X.reset_index(drop=True)], axis=1)
    out.attrs["normalisation"] = "log2(CPM+1)" if looks_like_counts else "z-score"
    return out


# ---------------------------------------------------------------- differential test
def differential(dfn: pd.DataFrame, group_col: str, reference: str | None = None) -> pd.DataFrame:
    groups = list(dict.fromkeys(dfn[group_col]))
    if len(groups) != 2:
        raise ValueError(f"differential() needs exactly 2 groups, found {groups}")
    ref = reference or groups[0]
    trt = [g for g in groups if g != ref][0]
    X = feature_matrix(dfn, group_col)
    a, b = X[dfn[group_col] == ref], X[dfn[group_col] == trt]
    t, p = stats.ttest_ind(b, a, equal_var=False, nan_policy="omit")   # Welch t-test per feature
    p = np.nan_to_num(p, nan=1.0)
    _, fdr, _, _ = multipletests(p, method="fdr_bh")
    res = pd.DataFrame({
        "feature": X.columns,
        "mean_" + ref: a.mean().values,
        "mean_" + trt: b.mean().values,
        "log2FC": (b.mean() - a.mean()).values,          # already log scale after normalisation
        "t": t, "p_value": p, "fdr": fdr,
    }).sort_values("fdr")
    res.attrs.update({"reference": ref, "treatment": trt})
    return res


# ---------------------------------------------------------------------------- figures
def plot_volcano(res: pd.DataFrame, out: Path, fdr_cut: float = 0.05, fc_cut: float = 1.0) -> Path:
    sig = (res["fdr"] < fdr_cut) & (res["log2FC"].abs() >= fc_cut)
    plt.figure(figsize=(6, 5))
    plt.scatter(res["log2FC"], -np.log10(res["p_value"].clip(1e-300)), s=10, c="#9aa5b1", label="n.s.")
    plt.scatter(res.loc[sig, "log2FC"], -np.log10(res.loc[sig, "p_value"].clip(1e-300)), s=12, c="#d1495b",
                label=f"FDR<{fdr_cut} & |log2FC|≥{fc_cut} (n={int(sig.sum())})")
    plt.axvline(-fc_cut, ls="--", lw=0.8, c="k"); plt.axvline(fc_cut, ls="--", lw=0.8, c="k")
    plt.xlabel("log2 fold change"); plt.ylabel("-log10 p-value"); plt.legend(fontsize=8)
    plt.title(f"Volcano: {res.attrs.get('treatment')} vs {res.attrs.get('reference')}")
    plt.tight_layout(); out.parent.mkdir(parents=True, exist_ok=True); plt.savefig(out, dpi=150); plt.close()
    return out


def plot_pca(dfn: pd.DataFrame, group_col: str, out: Path) -> Path:
```

```
    X = feature_matrix(dfn, group_col)
    pcs = PCA(n_components=2).fit(X)
    Z = pcs.transform(X)
    plt.figure(figsize=(5.5, 5))
    for g in dfn[group_col].unique():
        m = dfn[group_col].values == g
        plt.scatter(Z[m, 0], Z[m, 1], label=str(g), s=60)
    for i, sid in enumerate(dfn["sample_id"]):
        plt.annotate(str(sid), (Z[i, 0], Z[i, 1]), fontsize=7, alpha=0.7)
    plt.xlabel(f"PC1 ({pcs.explained_variance_ratio_[0]:.0%})"); plt.ylabel(f"PC2 ({pcs.explained_variance_ratio_
↪   [1]:.0%})")
    plt.legend(); plt.title("PCA of samples"); plt.tight_layout(); plt.savefig(out, dpi=150); plt.close()
    return out


# ---------------------------------------------------------------------------- ML
def machine_learning(dfn: pd.DataFrame, group_col: str, top_k: int = 50, seed: int = 0) -> dict:
    """Baseline classifiers with proper cross-validation (LOOCV when n is tiny)."""
    X = feature_matrix(dfn, group_col)
    X = X.loc[:, X.var(axis=0).sort_values(ascending=False).index[:top_k]]  # unsupervised filter → no leakage
    y = dfn[group_col].astype("category").cat.codes.values
    n_min = int(np.bincount(y).min())
    cv = LeaveOneOut() if n_min < 5 else StratifiedKFold(n_splits=min(5, n_min), shuffle=True, random_state=seed)
    cv_name = "leave-one-out" if isinstance(cv, LeaveOneOut) else f"stratified {cv.get_n_splits()}-fold"
    models = {
        "logistic_regression": make_pipeline(StandardScaler(), LogisticRegression(max_iter=2000, C=0.5)),
        "random_forest": RandomForestClassifier(n_estimators=300, random_state=seed),
    }
    results = {}
    for name, model in models.items():
        acc = cross_val_score(model, X.values, y, cv=cv, scoring="accuracy")
        results[name] = {"cv_accuracy_mean": float(acc.mean()), "cv_accuracy_std": float(acc.std())}
    rf = models["random_forest"].fit(X.values, y)
    imp = pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False)
    return {"cross_validation": cv_name, "n_features_used": int(X.shape[1]), "models": results,
            "top_features": imp.head(10).round(4).to_dict(),
            "warning": "Tiny sample size: accuracy estimates are high-variance" if len(y) < 12 else ""}


# ---------------------------------------------------------------------------- summary
def summarise(qc: dict, dfn: pd.DataFrame, res: pd.DataFrame, ml: dict, fdr_cut: float = 0.05) -> dict:
    sig = res[res["fdr"] < fdr_cut]
    return {
        "qc": qc,
        "normalisation": dfn.attrs.get("normalisation"),
        "comparison": f"{res.attrs.get('treatment')} vs {res.attrs.get('reference')}",
        "test": "Welch two-sample t-test per feature; Benjamini-Hochberg FDR",
        "n_features_tested": int(len(res)),
        "n_significant_fdr": int(len(sig)),
        "n_up": int((sig["log2FC"] > 0).sum()), "n_down": int((sig["log2FC"] < 0).sum()),
        "top_features": res.head(10)[["feature", "log2FC", "p_value", "fdr"]].round(4).to_dict("records"),
        "machine_learning": ml,
    }
```

18.2.3 `data.py`

```
"""
data.py — load a real CSV or generate a synthetic omics-style demo dataset.

Expected layout (samples as rows):
    sample_id, <group_col>, feature_1, feature_2, ...
e.g. an RNA-seq count table transposed so each row is a sample and each column a gene.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from pathlib import Path

import numpy as np
import pandas as pd


def make_demo(n_per_group: int = 3, n_features: int = 300, frac_de: float = 0.1,
              groups: tuple[str, str] = ("control", "heat"), seed: int = 42) -> pd.DataFrame:
    """Synthetic RNA-seq-like counts: negative-binomial-ish, 10% of genes differentially expressed."""
    rng = np.random.default_rng(seed)
    base = rng.gamma(shape=2.0, scale=200.0, size=n_features)          # gene-specific mean expression
    n_de = int(n_features * frac_de)
    effect = np.ones(n_features)
    de_idx = rng.choice(n_features, n_de, replace=False)
    effect[de_idx] = rng.choice([0.15, 0.25, 4.0, 6.0], size=n_de)       # down or up in treatment
    rows, ids, labels = [], [], []
    for g_i, g in enumerate(groups):
        for r in range(n_per_group):
            mu = base * (effect if g_i == 1 else 1.0)
            counts = rng.poisson(rng.gamma(shape=25.0, scale=mu / 25.0))   # over-dispersed counts
            rows.append(counts)
            ids.append(f"{g}_{r + 1}")
            labels.append(g)
    df = pd.DataFrame(rows, columns=[f"gene_{i + 1}" for i in range(n_features)])
    df.insert(0, "condition", labels)
    df.insert(0, "sample_id", ids)
    return df


def load_csv(path: Path, group_col: str, id_col: str | None = None) -> pd.DataFrame:
    df = pd.read_csv(path)
    if group_col not in df.columns:
        raise ValueError(f"group column '{group_col}' not in CSV columns: {list(df.columns)[:10]}...")
    if id_col and id_col in df.columns:
        df = df.rename(columns={id_col: "sample_id"})
    if "sample_id" not in df.columns:
        df.insert(0, "sample_id", [f"s{i + 1}" for i in range(len(df))])
    return df
```

18.2.4 `llm.py`

```
"""
llm.py — a thin wrapper around the Anthropic Claude API used by this agent.

Two functions are all you need:
    ask(system, user)                 -> free text (streamed)
    ask_structured(system, user, Model) -> a validated Pydantic object (JSON structured output)

Why a wrapper?  Every agent in this project talks to the model the same way.  Keeping the
API details in ONE file per agent means you can swap the model or provider in one place.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import os
from typing import Type, TypeVar

import anthropic
from dotenv import load_dotenv
from pydantic import BaseModel

load_dotenv()  # reads ANTHROPIC_API_KEY / CLAUDE_MODEL from a .env file if present

MODEL = os.getenv("CLAUDE_MODEL", "claude-opus-5")
T = TypeVar("T", bound=BaseModel)

_client: anthropic.Anthropic | None = None


def client() -> anthropic.Anthropic:
    """Lazily create one shared client (credentials come from the environment)."""
    global _client
    if _client is None:
        _client = anthropic.Anthropic()
    return _client


def _check(msg: anthropic.types.Message) -> None:
    if msg.stop_reason == "refusal":
        detail = getattr(msg, "stop_details", None)
        raise RuntimeError(f"The model refused this request: {detail}")
    if msg.stop_reason == "max_tokens":
        raise RuntimeError("Output was cut off (max_tokens). Increase max_tokens and retry.")


def _text(msg: anthropic.types.Message) -> str:
    return "".join(b.text for b in msg.content if b.type == "text")


def ask(system: str, user: str, *, max_tokens: int = 16000, effort: str = "medium",
        echo: bool = False) -> str:
    """Free-text generation. Streams so long outputs never hit HTTP timeouts."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
```

```
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
    ) as stream:
        if echo:
            for chunk in stream.text_stream:
                print(chunk, end="", flush=True)
            print()
        msg = stream.get_final_message()
    _check(msg)
    return _text(msg)


def ask_structured(system: str, user: str, model_cls: Type[T], *, max_tokens: int = 16000,
                   effort: str = "medium") -> T:
    """Structured generation: the SDK converts the Pydantic model to a JSON schema, the API
    constrains the output to it, and `parsed_output` is a validated instance of `model_cls`."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
        output_format=model_cls,
    ) as stream:
        msg = stream.get_final_message()
    _check(msg)
    if msg.parsed_output is None:  # should not happen with a constrained schema; be defensive
        return model_cls.model_validate_json(_text(msg))
    return msg.parsed_output
```

18.2.5 `main.py`

```
"""
AI Research Experiment Agent — "Virtual Research Assistant"
============================================================
description (+ data) → design → QC/normalise/test/ML (code) → statistics review → Methods & Results + notebook

Usage
    python main.py --description "RNA-seq of heat-stressed rice, control vs heat, n=3" --demo
    python main.py --description "..." --data mydata.csv --group-col condition
    python main.py --description "..." --demo --no-llm        # run only the numeric pipeline (no API key needed)

Outputs: outputs/<slug>/ design.json, data.csv, summary.json, differential.csv, pca.png, volcano.png,
         stats_review.json, methods_results.md, analysis_notebook.ipynb

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import json
import sys
from pathlib import Path
```

```
import typer
from rich.console import Console
from rich.panel import Panel
from rich.table import Table
from slugify import slugify

sys.path.insert(0, str(Path(__file__).parent))
if hasattr(sys.stdout, "reconfigure"):  # Windows consoles default to cp1252; we print → ✓ etc.
    sys.stdout.reconfigure(encoding="utf-8")
from analysis import (differential, machine_learning, normalise, plot_pca,  # noqa: E402
                      plot_volcano, quality_control, summarise)
from data import load_csv, make_demo  # noqa: E402
from notebook import build_notebook  # noqa: E402

app = typer.Typer(add_completion=False)
console = Console()
AGENT_DIR = Path(__file__).parent
OUT_DIR = AGENT_DIR / "outputs"

BANNER = """[bold cyan]AI Research Experiment Agent[/] — Virtual Research Assistant
[dim]Codanics · Python ka Chilla: Build AI Agents · https://codanics.com/courses/python-ka-chilla-build-ai-agents/
YouTube @codanics · WhatsApp https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r[/]"""


@app.command()
def run(
    description: str = typer.Option(..., help="One or two sentences describing the experiment/data"),
    data: Path = typer.Option(None, help="CSV with samples as rows (see data.py)"),
    group_col: str = typer.Option("condition", help="Column holding the group / treatment label"),
    demo: bool = typer.Option(False, help="Generate synthetic RNA-seq-like demo data"),
    n_per_group: int = typer.Option(3), n_features: int = typer.Option(300),
    no_llm: bool = typer.Option(False, help="Skip the LLM steps (numeric pipeline + notebook only)"),
    echo: bool = typer.Option(True),
):
    console.print(Panel(BANNER, expand=False))
    out = OUT_DIR / slugify(description)[:60]
    out.mkdir(parents=True, exist_ok=True)

    # 1) Data ---------------------------------------------------------------
    console.rule("[bold]1. Data")
    if demo or data is None:
        df = make_demo(n_per_group=n_per_group, n_features=n_features)
        console.print(f"  synthetic demo dataset: {df.shape[0]} samples × {n_features} features")
    else:
        df = load_csv(data, group_col)
        console.print(f"  loaded {data}: {df.shape}")
    csv_path = out / "data.csv"
    df.to_csv(csv_path, index=False)
    data_hint = (f"Data available: {df.shape[0]} samples, {df.shape[1] - 2} numeric features, "
                 f"groups = {df[group_col].value_counts().to_dict()}")

    # 2) Design (LLM) -------------------------------------------------------
    design = None
    if not no_llm:
        console.rule("[bold]2. Experiment design")
        from agents import ExperimentPlanner
        design = ExperimentPlanner().run(description, data_hint)
        console.print(f"  [bold]{design.title}[/]\n  H: {design.hypothesis}\n  tests: {design.statistical_tests}")
```

```
        (out / "design.json").write_text(design.model_dump_json(indent=2), encoding="utf-8")

    # 3) Analysis (code) ------------------------------------------------------
    console.rule("[bold]3. Analysis engine (pandas · scipy · statsmodels · scikit-learn)")
    qc = quality_control(df, group_col)
    dfn = normalise(df, group_col)
    res = differential(dfn, group_col)
    res.to_csv(out / "differential.csv", index=False)
    plot_pca(dfn, group_col, out / "pca.png")
    plot_volcano(res, out / "volcano.png")
    ml = machine_learning(dfn, group_col)
    summary = summarise(qc, dfn, res, ml)
    (out / "summary.json").write_text(json.dumps(summary, indent=2), encoding="utf-8")

    console.print(f"  normalisation: {summary['normalisation']}   comparison: {summary['comparison']}")
    console.print(f"  significant features (FDR<0.05): {summary['n_significant_fdr']} "
                  f"(up {summary['n_up']}, down {summary['n_down']})")
    t = Table(title="Top differential features")
    for c in ("feature", "log2FC", "p_value", "fdr"):
        t.add_column(c)
    for r in summary["top_features"]:
        t.add_row(str(r["feature"]), f"{r['log2FC']:.2f}", f"{r['p_value']:.2e}", f"{r['fdr']:.2e}")
    console.print(t)
    for name, m in ml["models"].items():
        console.print(f"  {name:<20} CV accuracy {m['cv_accuracy_mean']:.2f} ± {m['cv_accuracy_std']:.2f}  ({ml['cross_
↪ validation']})")
    if qc["warnings"]:
        console.print(f"  [yellow]QC warnings:[/] {qc['warnings']}")

    # 4) Notebook (code) ------------------------------------------------------
    console.rule("[bold]4. Reproducible notebook")
    nb = build_notebook(csv_path, group_col, AGENT_DIR, out / "analysis_notebook.ipynb", description)
    console.print(f"  [green]saved[/] {nb.name}")

    if no_llm:
        console.print(Panel(f"Numeric pipeline done (LLM skipped). Outputs in {out}", style="green"))
        return

    # 5) Statistics review + report (LLM) -------------------------------------
    console.rule("[bold]5. Statistics review")
    from agents import ReportWriter, StatisticsAgent
    review = StatisticsAgent().run(design, summary)
    console.print(f"  verdict: {review.overall_verdict}")
    for c in review.concerns:
        console.print(f"   - [{c.severity}] {c.issue} → {c.recommendation}")
    (out / "stats_review.json").write_text(review.model_dump_json(indent=2), encoding="utf-8")

    console.rule("[bold]6. Methods & Results")
    report = ReportWriter().run(description, design, summary, review, echo=echo)
    (out / "methods_results.md").write_text(report, encoding="utf-8")
    console.print(Panel(f"Done. Outputs in [bold]{out}[/]", style="green"))


if __name__ == "__main__":
    app()
```

18.2.6 `notebook.py`

```
"""
notebook.py — generate a runnable Jupyter notebook that reproduces the analysis.

Reproducibility is a product feature: the researcher can open the notebook, change a threshold,
and re-run everything without the agent.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from pathlib import Path

import nbformat as nbf


def build_notebook(csv_path: Path, group_col: str, agent_dir: Path, out: Path, title: str) -> Path:
    nb = nbf.v4.new_notebook()
    cells = [
        nbf.v4.new_markdown_cell(
            f"# {title}\n\nReproducible analysis notebook generated by the **AI Research Experiment Agent**.\n\n"
            f"Codanics — Python ka Chilla: Build AI Agents · "
            f"https://codanics.com/courses/python-ka-chilla-build-ai-agents/"),
        nbf.v4.new_code_cell(
            "import sys, json\nfrom pathlib import Path\nimport pandas as pd\n"
            f"sys.path.insert(0, r'{agent_dir}')\n"
            "from analysis import quality_control, normalise, differential, plot_pca, plot_volcano, machine_learning,
            ↪  summarise\n"
            f"CSV = Path(r'{csv_path}')\nGROUP = '{group_col}'\n"
            "df = pd.read_csv(CSV)\ndf.head()"),
        nbf.v4.new_markdown_cell("## 1. Quality control"),
        nbf.v4.new_code_cell("qc = quality_control(df, GROUP)\nqc"),
        nbf.v4.new_markdown_cell("## 2. Normalisation and PCA"),
        nbf.v4.new_code_cell("dfn = normalise(df, GROUP)\nprint(dfn.attrs['normalisation'])\n"
                             "plot_pca(dfn, GROUP, Path('pca.png'))\nfrom IPython.display import Image;
                             ↪  Image('pca.png')"),
        nbf.v4.new_markdown_cell("## 3. Differential analysis (Welch t-test + BH-FDR)"),
        nbf.v4.new_code_cell("res = differential(dfn, GROUP)\nres.head(15)"),
        nbf.v4.new_code_cell("plot_volcano(res, Path('volcano.png'))\nImage('volcano.png')"),
        nbf.v4.new_markdown_cell("## 4. Machine-learning baseline with cross-validation"),
        nbf.v4.new_code_cell("ml = machine_learning(dfn, GROUP)\nml"),
        nbf.v4.new_markdown_cell("## 5. Summary passed to the Statistics Agent"),
        nbf.v4.new_code_cell("print(json.dumps(summarise(qc, dfn, res, ml), indent=1))"),
    ]
    nb["cells"] = cells
    out.parent.mkdir(parents=True, exist_ok=True)
    nbf.write(nb, out)
    return out
```

19 Code: Agent 4 — Academic Writing & Publication

Folder: `02_agents_code/agent_04_academic_writing/` — self-contained; run `python agent_04_academic_writing/main.py --help`.

19.1 README

19.1.1 `README.md`

```
# Agent 4 — AI Academic Writing & Publication Agent ("From Data to Manuscript")

A multi-agent publishing assistant: **manuscript drafting → journal matching → simulated peer review (3 personas) →
↪  response letter + revised manuscript**.

```
notes.md ─► ManuscriptAgent (IMRaD draft + AI-use declaration)
         ─► JournalMatchingAgent (scores journals from journals.py catalogue)
         ─► ReviewerSimulationAgent (methodologist · domain expert · statistician)
         ─► RevisionAgent (point-by-point response letter, revised manuscript with bold changes)
```

Files

File	Purpose
`main.py`	CLI (`--notes`, `--manuscript`, `--references`, `--field`, `--reviewers`)
`agents.py`	The four sub-agents + Pydantic models (JournalRanking, ReviewSet)
`journals.py`	Editable journal catalogue (replace with OpenAlex/Scimago/DOAJ in production)
`sample_notes.md`	Example study notes to try immediately
`llm.py`	Thin Claude client
`outputs/<slug>/`	`manuscript_v1.md`, `journal_matches.json`, `reviews.json`, `response_letter.md`, `manuscript_v2.md`
↪

Run

```powershell
python agent_04_academic_writing\main.py --notes agent_04_academic_writing\sample_notes.md --field "plant science"
python agent_04_academic_writing\main.py --manuscript my_paper.md --field "medicine" --reviewers 2
```

What this agent teaches

1. **Multi-persona simulation** — the same model plays three reviewers with different priorities.
2. **Critique → revision loops** — the classic "generator + critic" agent pattern.
3. **Guardrails against fabrication** — `[TO BE ADDED BY AUTHORS]` instead of invented facts; AI-use declaration appended
↪ automatically.
4. **Knowledge bases in code** — a small journal catalogue the model must choose from.
```

```
## Ideas to extend

- Live journal data (OpenAlex, Scimago), predatory-journal blacklist, Word/LaTeX export per journal template,
↪  Zotero/BibTeX, grammar pass, cover-letter generator.

---

**Codanics — Python ka Chilla: Build AI Agents**
Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/ · Website: https://codanics.com · YouTube:
↪  [@codanics](https://www.youtube.com/@codanics)
WhatsApp: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r · Free course:
↪  https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40
```

19.2 Source files

19.2.1 agents.py

```
"""
agents.py — the four sub-agents of the Academic Writing & Publication Agent.

    ManuscriptAgent       : notes/results → IMRaD manuscript (Markdown)
    JournalMatchingAgent  : manuscript + catalogue → ranked journals (structured)
    ReviewerSimulationAgent: manuscript → N reviewers with personas (structured)
    RevisionAgent         : manuscript + reviews → response letter + revised manuscript

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from pydantic import BaseModel, Field

from journals import catalogue_text
from llm import ask, ask_structured

BRAND = ("Codanics — Python ka Chilla: Build AI Agents | "
         "https://codanics.com/courses/python-ka-chilla-build-ai-agents/")

NO_FABRICATION = ("STRICT RULE: use only facts, numbers and references present in the input. If something is "
                  "missing, write '[TO BE ADDED BY AUTHORS]' instead of inventing it.")

AI_STATEMENT = ("\n\n## Declaration of AI assistance\n"
                "An AI writing assistant (AI Academic Writing & Publication Agent) was used to draft and structure "
                "this manuscript from the authors' notes and results. All content was reviewed, verified and "
                "approved by the authors, who take full responsibility for the work.\n")


# ---------------------------------------------------------------- 1. Manuscript
class ManuscriptAgent:
    SYSTEM = ("You are an experienced academic writer and editor. Write a complete manuscript in Markdown "
              "with: Title; Authors placeholder; Abstract (250 words, structured); Keywords; 1 Introduction "
              "(with the gap and objectives); 2 Materials and Methods; 3 Results; 4 Discussion; 5 Conclusion; "
              "Acknowledgements placeholder; References (only those given). Clear, concise academic English "
              "suitable for non-native authors. " + NO_FABRICATION)
```

```
    def run(self, notes: str, field: str, references: str = "", echo: bool = False) -> str:
        text = ask(self.SYSTEM,
                   f"Field: {field}\n\nAuthor notes and results:\n{notes}\n\nReferences available:\n"
                   f"{references or '(none provided)'}\n\nWrite the manuscript.",
                   max_tokens=32000, effort="high", echo=echo)
        return text + AI_STATEMENT


# ---------------------------------------------------------- 2. Journal matching
class JournalMatch(BaseModel):
    journal: str
    scope_fit: int = Field(ge=1, le=10)
    impact_band: str
    open_access: str
    estimated_apc_usd: int | None
    acceptance_probability: str = Field(description="low | medium | high, with the main reason")
    reason: str
    tips_for_this_journal: str


class JournalRanking(BaseModel):
    manuscript_summary: str
    matches: list[JournalMatch] = Field(description="Top 5, best first")
    avoid: list[str] = Field(description="Journals from the catalogue that are a poor fit, with a reason")


class JournalMatchingAgent:
    SYSTEM = ("You are a journal-selection consultant. Score journals from the provided catalogue ONLY "
              "(never invent journals). Balance scope fit, impact, cost, speed and realistic acceptance odds.")

    def run(self, manuscript: str) -> JournalRanking:
        return ask_structured(self.SYSTEM,
                              f"Journal catalogue:\n{catalogue_text()}\n\nManuscript:\n{manuscript[:20000]}",
                              JournalRanking, effort="medium")


# ---------------------------------------------------------- 3. Reviewer simulation
class ReviewComment(BaseModel):
    number: int
    section: str
    comment: str
    severity: str = Field(description="minor | major | critical")


class Review(BaseModel):
    reviewer: str = Field(description="e.g. 'Reviewer 1 (methodologist)'")
    summary: str
    comments: list[ReviewComment]
    recommendation: str = Field(description="accept | minor revision | major revision | reject")


class ReviewSet(BaseModel):
    reviews: list[Review]


PERSONAS = [
    "Reviewer 1 — a strict methodologist who focuses on validation, controls, sample size and reproducibility",
    "Reviewer 2 — a domain expert who checks novelty, dataset selection, related work and interpretation",
```

```
    "Reviewer 3 — a statistician/data scientist who checks tests, leakage, metrics and figures",
]


class ReviewerSimulationAgent:
    SYSTEM = ("You simulate peer review for a scientific manuscript. Each reviewer has a distinct persona and "
              "must give specific, numbered, actionable comments (quote the problematic sentence when possible). "
              "Be tough but fair.")

    def run(self, manuscript: str, n_reviewers: int = 3) -> ReviewSet:
        personas = "\n".join(PERSONAS[:n_reviewers])
        return ask_structured(self.SYSTEM,
                              f"Reviewer personas:\n{personas}\n\nManuscript:\n{manuscript[:30000]}\n\n"
                              f"Produce {n_reviewers} reviews.",
                              ReviewSet, effort="high")


# ---------------------------------------------------------------- 4. Revision
class RevisionAgent:
    SYSTEM_LETTER = ("You write point-by-point response letters to reviewers. For EVERY comment: quote it, "
                     "respond politely and concretely, and state exactly what was changed and where "
                     "(section/paragraph). If a change is impossible, explain why respectfully. Markdown.")
    SYSTEM_REVISE = ("You revise scientific manuscripts to address reviewer comments. Return the FULL revised "
                     "manuscript in Markdown. Mark changed passages by wrapping them in **bold**. "
                     + NO_FABRICATION)

    def letter(self, manuscript: str, reviews: ReviewSet, echo: bool = False) -> str:
        text = ask(self.SYSTEM_LETTER,
                   f"Reviews:\n{reviews.model_dump_json(indent=1)}\n\nManuscript:\n{manuscript[:30000]}",
                   effort="high", echo=echo)
        return text + f"\n\n---\n*Generated with the AI Academic Writing & Publication Agent — {BRAND}*\n"

    def revise(self, manuscript: str, reviews: ReviewSet, echo: bool = False) -> str:
        return ask(self.SYSTEM_REVISE,
                   f"Reviews:\n{reviews.model_dump_json(indent=1)}\n\nOriginal manuscript:\n{manuscript}",
                   max_tokens=32000, effort="high", echo=echo)
```

19.2.2 `journals.py`

```
"""
journals.py — a small, editable journal knowledge base for the Journal Matching Agent.

In production replace this with live data (OpenAlex / Scimago / DOAJ APIs) and a predatory-journal
blacklist.  Impact bands are indicative only.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from pydantic import BaseModel


class Journal(BaseModel):
    name: str
    publisher: str
```

```
    scope: str
    fields: list[str]
    impact_band: str          # "very high" | "high" | "medium" | "emerging"
    open_access: str          # "gold" | "hybrid" | "subscription"
    apc_usd: int | None       # article processing charge if OA
    review_weeks: int         # typical first-decision time
    acceptance_rate: str      # rough band


JOURNALS: list[Journal] = [
    Journal(name="Computers and Electronics in Agriculture", publisher="Elsevier",
            scope="Computing, sensing and AI applied to agriculture and food production",
            fields=["agriculture", "plant science", "AI", "remote sensing"], impact_band="high",
            open_access="hybrid", apc_usd=3500, review_weeks=8, acceptance_rate="~25%"),
    Journal(name="Plant Phenomics", publisher="Science Partner Journals / AAAS",
            scope="High-throughput plant phenotyping, imaging, sensors and analytics",
            fields=["plant science", "phenotyping", "AI"], impact_band="high",
            open_access="gold", apc_usd=3000, review_weeks=6, acceptance_rate="~30%"),
    Journal(name="Frontiers in Plant Science", publisher="Frontiers",
            scope="All areas of plant biology including stress physiology and computational approaches",
            fields=["plant science", "biology", "stress"], impact_band="medium",
            open_access="gold", apc_usd=3300, review_weeks=8, acceptance_rate="~50%"),
    Journal(name="Remote Sensing", publisher="MDPI",
            scope="Remote-sensing science and applications including UAV and hyperspectral imaging",
            fields=["remote sensing", "agriculture", "AI"], impact_band="medium",
            open_access="gold", apc_usd=2700, review_weeks=4, acceptance_rate="~45%"),
    Journal(name="Scientific Reports", publisher="Springer Nature",
            scope="Multidisciplinary, technically sound research in natural sciences",
            fields=["multidisciplinary", "biology", "AI", "medicine"], impact_band="medium",
            open_access="gold", apc_usd=2690, review_weeks=10, acceptance_rate="~45%"),
    Journal(name="Nature Machine Intelligence", publisher="Springer Nature",
            scope="High-impact AI/ML research with broad scientific significance",
            fields=["AI", "machine learning", "multidisciplinary"], impact_band="very high",
            open_access="hybrid", apc_usd=12000, review_weeks=12, acceptance_rate="<10%"),
    Journal(name="Bioinformatics", publisher="Oxford University Press",
            scope="Computational methods and tools for biological data analysis",
            fields=["bioinformatics", "genomics", "biology"], impact_band="high",
            open_access="hybrid", apc_usd=3500, review_weeks=8, acceptance_rate="~20%"),
    Journal(name="BMC Bioinformatics", publisher="Springer Nature",
            scope="Bioinformatics methods, databases and applications",
            fields=["bioinformatics", "genomics"], impact_band="medium",
            open_access="gold", apc_usd=2790, review_weeks=10, acceptance_rate="~35%"),
    Journal(name="Theoretical and Applied Genetics", publisher="Springer",
            scope="Plant genetics, genomics and breeding",
            fields=["genetics", "plant science", "breeding"], impact_band="high",
            open_access="hybrid", apc_usd=3690, review_weeks=8, acceptance_rate="~25%"),
    Journal(name="Journal of Medical Internet Research", publisher="JMIR Publications",
            scope="Digital health, AI in medicine, eHealth",
            fields=["medicine", "digital health", "AI"], impact_band="high",
            open_access="gold", apc_usd=3300, review_weeks=6, acceptance_rate="~25%"),
    Journal(name="IEEE Access", publisher="IEEE",
            scope="Multidisciplinary engineering and computer science, rapid publication",
            fields=["engineering", "computer science", "AI"], impact_band="medium",
            open_access="gold", apc_usd=1995, review_weeks=4, acceptance_rate="~30%"),
    Journal(name="PLOS ONE", publisher="PLOS",
            scope="Multidisciplinary; rigorous methodology rather than perceived impact",
            fields=["multidisciplinary", "biology", "medicine", "social science"], impact_band="medium",
            open_access="gold", apc_usd=2290, review_weeks=10, acceptance_rate="~45%"),
    Journal(name="Expert Systems with Applications", publisher="Elsevier",
```

```
            scope="Applied AI and intelligent systems across domains",
            fields=["AI", "computer science", "applications"], impact_band="high",
            open_access="hybrid", apc_usd=3600, review_weeks=10, acceptance_rate="~20%"),
    Journal(name="Agricultural Water Management", publisher="Elsevier",
            scope="Water use, irrigation, drought and crop-water relations",
            fields=["agriculture", "water", "drought"], impact_band="high",
            open_access="hybrid", apc_usd=3500, review_weeks=8, acceptance_rate="~25%"),
    Journal(name="Heliyon", publisher="Cell Press / Elsevier",
            scope="Multidisciplinary open-access journal for technically sound research",
            fields=["multidisciplinary"], impact_band="emerging",
            open_access="gold", apc_usd=2000, review_weeks=6, acceptance_rate="~50%"),
]


def catalogue_text() -> str:
    return "\n".join(j.model_dump_json() for j in JOURNALS)
```

19.2.3 `llm.py`

```
"""
llm.py — a thin wrapper around the Anthropic Claude API used by this agent.

Two functions are all you need:
    ask(system, user)                  -> free text (streamed)
    ask_structured(system, user, Model) -> a validated Pydantic object (JSON structured output)

Why a wrapper?  Every agent in this project talks to the model the same way.  Keeping the
API details in ONE file per agent means you can swap the model or provider in one place.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import os
from typing import Type, TypeVar

import anthropic
from dotenv import load_dotenv
from pydantic import BaseModel

load_dotenv()  # reads ANTHROPIC_API_KEY / CLAUDE_MODEL from a .env file if present

MODEL = os.getenv("CLAUDE_MODEL", "claude-opus-5")
T = TypeVar("T", bound=BaseModel)

_client: anthropic.Anthropic | None = None


def client() -> anthropic.Anthropic:
    """Lazily create one shared client (credentials come from the environment)."""
    global _client
    if _client is None:
        _client = anthropic.Anthropic()
    return _client
```

```python
def _check(msg: anthropic.types.Message) -> None:
    if msg.stop_reason == "refusal":
        detail = getattr(msg, "stop_details", None)
        raise RuntimeError(f"The model refused this request: {detail}")
    if msg.stop_reason == "max_tokens":
        raise RuntimeError("Output was cut off (max_tokens). Increase max_tokens and retry.")


def _text(msg: anthropic.types.Message) -> str:
    return "".join(b.text for b in msg.content if b.type == "text")


def ask(system: str, user: str, *, max_tokens: int = 16000, effort: str = "medium",
        echo: bool = False) -> str:
    """Free-text generation. Streams so long outputs never hit HTTP timeouts."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
    ) as stream:
        if echo:
            for chunk in stream.text_stream:
                print(chunk, end="", flush=True)
            print()
        msg = stream.get_final_message()
    _check(msg)
    return _text(msg)


def ask_structured(system: str, user: str, model_cls: Type[T], *, max_tokens: int = 16000,
                   effort: str = "medium") -> T:
    """Structured generation: the SDK converts the Pydantic model to a JSON schema, the API
    constrains the output to it, and `parsed_output` is a validated instance of `model_cls`."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
        output_format=model_cls,
    ) as stream:
        msg = stream.get_final_message()
    _check(msg)
    if msg.parsed_output is None:  # should not happen with a constrained schema; be defensive
        return model_cls.model_validate_json(_text(msg))
    return msg.parsed_output
```

19.2.4 `main.py`

```
"""
AI Academic Writing & Publication Agent — "From Data to Manuscript"
```

```
================================================================
notes -> manuscript -> journal matching -> reviewer simulation -> response letter + revised manuscript

Usage
    python main.py --notes sample_notes.md --field "plant science"
    python main.py --notes my_notes.md --field "medicine" --reviewers 2 --references refs.md
    python main.py --manuscript existing_paper.md --field "AI"      # skip drafting; review an existing paper

Outputs: outputs/<slug>/ manuscript_v1.md, journal_matches.json, reviews.json, response_letter.md, manuscript_v2.md

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import sys
from pathlib import Path

import typer
from rich.console import Console
from rich.panel import Panel
from rich.table import Table
from slugify import slugify

sys.path.insert(0, str(Path(__file__).parent))
if hasattr(sys.stdout, "reconfigure"):  # Windows consoles default to cp1252; we print → ✓ etc.
    sys.stdout.reconfigure(encoding="utf-8")
from agents import (JournalMatchingAgent, ManuscriptAgent, ReviewerSimulationAgent,  # noqa: E402
                    RevisionAgent)

app = typer.Typer(add_completion=False)
console = Console()
OUT_DIR = Path(__file__).parent / "outputs"

BANNER = """[bold cyan]AI Academic Writing & Publication Agent[/] — From Data to Manuscript
[dim]Codanics · Python ka Chilla: Build AI Agents · https://codanics.com/courses/python-ka-chilla-build-ai-agents/
YouTube @codanics · WhatsApp https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r[/]"""


def save(path: Path, text: str) -> None:
    path.parent.mkdir(parents=True, exist_ok=True)
    path.write_text(text, encoding="utf-8")
    console.print(f"  [green]saved[/] {path.name}")


@app.command()
def run(
    notes: Path = typer.Option(None, help="Markdown/text file with study notes and results"),
    manuscript: Path = typer.Option(None, help="Existing manuscript (.md) to review instead of drafting"),
    references: Path = typer.Option(None, help="Optional reference list file the writer may cite"),
    field: str = typer.Option("multidisciplinary", help="Research field, e.g. 'plant science'"),
    reviewers: int = typer.Option(3, min=1, max=3),
    echo: bool = typer.Option(True),
):
    console.print(Panel(BANNER, expand=False))
    if not notes and not manuscript:
        raise typer.BadParameter("Provide --notes or --manuscript")
    src = manuscript or notes
    out = OUT_DIR / slugify(src.stem)[:60]
```

```
    refs = references.read_text(encoding="utf-8") if references else ""

    # 1) Manuscript ------------------------------------------------------------
    console.rule("[bold]1. Manuscript")
    if manuscript:
        ms = manuscript.read_text(encoding="utf-8")
        console.print(f"  using existing manuscript {manuscript}")
    else:
        ms = ManuscriptAgent().run(notes.read_text(encoding="utf-8"), field, refs, echo=echo)
    save(out / "manuscript_v1.md", ms)

    # 2) Journal matching ------------------------------------------------------
    console.rule("[bold]2. Journal matching")
    ranking = JournalMatchingAgent().run(ms)
    t = Table(title="Recommended journals")
    for c in ("journal", "scope fit", "impact", "OA", "APC $", "acceptance"):
        t.add_column(c)
    for m in ranking.matches:
        t.add_row(m.journal, str(m.scope_fit), m.impact_band, m.open_access,
                  str(m.estimated_apc_usd or "-"), m.acceptance_probability)
    console.print(t)
    save(out / "journal_matches.json", ranking.model_dump_json(indent=2))

    # 3) Reviewer simulation ---------------------------------------------------
    console.rule(f"[bold]3. Simulated peer review ({reviewers} reviewers)")
    reviews = ReviewerSimulationAgent().run(ms, reviewers)
    for r in reviews.reviews:
        console.print(f"  [bold]{r.reviewer}[/] → {r.recommendation}")
        for c in r.comments[:4]:
            console.print(f"     {c.number}. [{c.severity}] {c.comment[:140]}")
    save(out / "reviews.json", reviews.model_dump_json(indent=2))

    # 4) Revision --------------------------------------------------------------
    console.rule("[bold]4. Response letter")
    rev = RevisionAgent()
    save(out / "response_letter.md", rev.letter(ms, reviews, echo=echo))
    console.rule("[bold]5. Revised manuscript")
    save(out / "manuscript_v2.md", rev.revise(ms, reviews, echo=False))
    console.print(Panel(f"Done. Outputs in [bold]{out}[/]", style="green"))


if __name__ == "__main__":
    app()
```

19.2.5 `sample_notes.md`

```
# Study notes — Multimodal AI for drought-stress prediction in wheat

## Aim
Predict drought stress in bread wheat (Triticum aestivum) 7 days before visible symptoms by fusing
UAV RGB imagery, canopy temperature and 2,000 SNP markers.

## Setup
- Field trial, Faisalabad, Pakistan, 2024–25 season.
- 120 genotypes × 2 water regimes (well-watered vs. drought from booting stage) × 3 replicates = 720 plots.
- UAV flights every 5 days (RGB + thermal); soil moisture sensors; SNP genotyping (2,000 markers).
```

- Ground truth: leaf relative water content (RWC) and visual stress score.

Models compared
- RGB-only CNN (ResNet-18)
- Thermal + weather gradient boosting
- Genomic-only GBLUP
- **Proposed**: late-fusion model (CNN embedding + thermal features + genomic PCs) with gradient boosting

Key results
- Proposed fusion model: accuracy 91.3%, F1 0.90, AUC 0.95 (5-fold CV, genotype-grouped folds).
- RGB-only: accuracy 84.1%; thermal+weather: 86.7%; genomic-only: 71.2%.
- Fusion detected stress on average 6.4 days before visible symptoms (vs 3.1 days RGB-only).
- SHAP: canopy temperature depression and two SNP clusters on chromosomes 2B and 5A were the most important features.
- Model trained on 2024 data generalised to 2025 with accuracy 87.9%.

Limitations noted by the team
- Single location; two seasons.
- Genotyping cost limits deployment for smallholders.
- Thermal imagery sensitive to time of day.

References we want to cite (authors' own list)
- Khan A, Ali B (2023). Deep learning for drought stress detection in wheat using RGB imagery. Computers and Electronics in Agriculture.
- Zhang C (2022). Hyperspectral indices for early drought detection in cereals: a review. Remote Sensing.
- Rehman D, Fatima E (2024). Genomic prediction of drought tolerance in wheat landraces. Theoretical and Applied Genetics.

20 Code: Agent 5 — PhD Supervisor

Folder: `02_agents_code/agent_05_phd_supervisor/` — self-contained; run `python agent_05_phd_supervisor/main.py --help`.

20.1 README

20.1.1 `README.md`

````
# Agent 5 — AI PhD Supervisor Agent ("Virtual Research Mentor")

A **long-running, memory-backed mentor**: it keeps a per-student roadmap and an append-only progress log, evaluates weekly
↪  updates against the plan, challenges the work like a committee member, and prepares slides/posters/defence Q&A.

```
init     → ResearchPlanningAgent  → memory/<student>/roadmap.json + outputs/<student>/roadmap.md
weekly   → WeeklyProgressAgent    → progress_log.jsonl + outputs/<student>/week_NN.md
critique → CriticalReviewerAgent  → critique_week_NN.md
present  → PresentationAgent      → slides.md | poster.md | defence.md
```

## Files

| File | Purpose |
|------|---------|
| `main.py` | Sub-commands: `init`, `weekly`, `critique`, `present`, `status` |
| `agents.py` | Roadmap (structured), weekly evaluation (structured), critic, presentation coach |
| `memory.py` | File-based memory: profile, roadmap, JSONL log, `context_summary()` for prompts |
| `llm.py` | Thin Claude client |
| `memory/` | One folder per student (audit trail the supervisor can read) |
| `outputs/` | Human-readable reports |

## Run

```powershell
python agent_05_phd_supervisor\main.py init --student "Ali" --topic "Multimodal AI for drought resilience in wheat"
↪  --years 3
python agent_05_phd_supervisor\main.py weekly --student "Ali" --update "Finished RGB pipeline, accuracy 87%, hyperspectral
↪  data delayed"
python agent_05_phd_supervisor\main.py critique --student "Ali"
python agent_05_phd_supervisor\main.py present --student "Ali" --type defence
python agent_05_phd_supervisor\main.py status --student "Ali"
```

## What this agent teaches

1. **Memory & state** — the agent is only as good as the context you give it; `context_summary()` sends roadmap + last N
↪  updates, not everything.
2. **Append-only logs** as an audit trail (and as supervision evidence for accreditation).
3. **Structured evaluation** — progress %, status, problems, goals: renderable in a dashboard later.
````

```
4. **Responsible AI framing** — every output carries a "supports, not replaces, your supervisor" notice.

## Ideas to extend

- Supervisor dashboard (Streamlit) across students with traffic-light status; reminders via WhatsApp/email; chapter
↪  review; mock viva with voice; cohort analytics.

---

**Codanics — Python ka Chilla: Build AI Agents**
Course: https://codanics.com/courses/python-ka-chilla-build-ai-agents/ · Website: https://codanics.com · YouTube:
↪  [@codanics](https://www.youtube.com/@codanics)
WhatsApp: https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r · Free course:
↪  https://www.youtube.com/watch?v=UE3o-7KRNXs&list=PLaQmdklCcX40
```

20.2 Source files

20.2.1 agents.py

```
"""
agents.py — the four sub-agents of the PhD Supervisor ("Virtual Research Mentor") Agent.

    ResearchPlanningAgent : profile → Roadmap (structured, milestones per semester)
    WeeklyProgressAgent   : memory + update → Evaluation (structured: %, status, problems, goals)
    CriticalReviewerAgent : memory → critique.md (assumptions, methodology, novelty)
    PresentationAgent     : memory → slides / poster / defence Q&A (Markdown)

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

from pydantic import BaseModel, Field

from llm import ask, ask_structured

BRAND = ("Codanics — Python ka Chilla: Build AI Agents | "
         "https://codanics.com/courses/python-ka-chilla-build-ai-agents/")

NOTICE = ("\n\n> **Note.** This feedback is generated by an AI research mentor to *support* — not replace — "
          "your supervisor. Verify important decisions with your supervisor and follow your "
          "institution's academic-integrity policy.\n")


# ---------------------------------------------------------------------- 1. Roadmap
class Milestone(BaseModel):
    semester: int = Field(description="1..N (two semesters per year)")
    title: str
    deliverables: list[str]
    success_criteria: str
    risks: list[str]


class Roadmap(BaseModel):
    programme_title: str
```

```
    aims: list[str] = Field(description="3-4 research aims")
    milestones: list[Milestone]
    publications_plan: list[str] = Field(description="Planned papers/conferences with target semester")
    skills_to_learn: list[str]
    weekly_routine: list[str] = Field(description="Recommended weekly habits, e.g. 'write 500 words'")


class ResearchPlanningAgent:
    SYSTEM = ("You are an experienced PhD supervisor and programme director. Create a realistic, "
              "milestone-based PhD roadmap with two semesters per year, front-loading literature review and "
              "methods, leaving the final year for writing and defence.")

    def run(self, name: str, topic: str, years: int, background: str = "") -> Roadmap:
        return ask_structured(self.SYSTEM,
                              f"Student: {name}\nTopic: {topic}\nProgramme length: {years} years\n"
                              f"Background: {background or 'not provided'}\nCreate the roadmap.",
                              Roadmap, effort="high")


# ------------------------------------------------------------ 2. Weekly progress
class Evaluation(BaseModel):
    progress_percent: int = Field(ge=0, le=100, description="Progress toward the current milestone")
    status: str = Field(description="on track | at risk | behind")
    achievements: list[str]
    problems: list[str] = Field(description="Specific issues, e.g. 'Need more validation experiments'")
    next_week_goals: list[str] = Field(description="3-5 concrete, verifiable goals")
    supervisor_should_know: str = Field(description="One line the human supervisor should see")
    encouragement: str


class WeeklyProgressAgent:
    SYSTEM = ("You are a supportive but rigorous research mentor doing a weekly check-in. Compare the update "
              "with the roadmap and recent history; detect drift, stalls and missing validation; set "
              "concrete goals. Be specific and kind.")

    def run(self, context: str, update: str) -> Evaluation:
        return ask_structured(self.SYSTEM, f"{context}\n\nTHIS WEEK'S UPDATE:\n{update}\n\nEvaluate.",
                              Evaluation, effort="medium")


# ------------------------------------------------------------ 3. Critical review
class CriticalReviewerAgent:
    SYSTEM = ("You are a tough but constructive thesis committee member. Challenge the research assumptions, "
              "methodology, novelty and feasibility based on the roadmap and progress so far. Output Markdown "
              "with sections: Assumptions to test; Methodological weaknesses; Novelty check; Questions the "
              "committee will ask; Recommended actions (prioritised).")

    def run(self, context: str, echo: bool = False) -> str:
        text = ask(self.SYSTEM, context, effort="high", echo=echo)
        return text + NOTICE + f"\n*Generated with the AI PhD Supervisor Agent — {BRAND}*\n"


# ------------------------------------------------------------ 4. Presentation
class PresentationAgent:
    SYSTEMS = {
        "slides": ("Create a 12-15 slide conference presentation outline in Markdown (one '## Slide N: title' "
                   "per slide with 3-5 bullet points and a speaker note)."),
        "poster": ("Create academic poster content in Markdown: title, authors placeholder, Introduction, "
                   "Methods, Results (describe figures), Conclusions, Future work, References placeholder."),
```

```
        "defence": ("Prepare the student for the PhD defence (viva): a 20-minute talk outline, 25 likely "
                    "committee questions grouped by theme with model answers, and 10 tips."),
    }

    def run(self, context: str, kind: str = "slides", echo: bool = False) -> str:
        system = "You are a presentation coach for researchers. " + self.SYSTEMS[kind]
        text = ask(system, context, effort="medium", echo=echo)
        return text + NOTICE + f"\n*Generated with the AI PhD Supervisor Agent — {BRAND}*\n"
```

20.2.2 `llm.py`

```
"""
llm.py — a thin wrapper around the Anthropic Claude API used by this agent.

Two functions are all you need:
    ask(system, user)                  -> free text (streamed)
    ask_structured(system, user, Model) -> a validated Pydantic object (JSON structured output)

Why a wrapper?  Every agent in this project talks to the model the same way.  Keeping the
API details in ONE file per agent means you can swap the model or provider in one place.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import os
from typing import Type, TypeVar

import anthropic
from dotenv import load_dotenv
from pydantic import BaseModel

load_dotenv()  # reads ANTHROPIC_API_KEY / CLAUDE_MODEL from a .env file if present

MODEL = os.getenv("CLAUDE_MODEL", "claude-opus-5")
T = TypeVar("T", bound=BaseModel)

_client: anthropic.Anthropic | None = None


def client() -> anthropic.Anthropic:
    """Lazily create one shared client (credentials come from the environment)."""
    global _client
    if _client is None:
        _client = anthropic.Anthropic()
    return _client


def _check(msg: anthropic.types.Message) -> None:
    if msg.stop_reason == "refusal":
        detail = getattr(msg, "stop_details", None)
        raise RuntimeError(f"The model refused this request: {detail}")
    if msg.stop_reason == "max_tokens":
        raise RuntimeError("Output was cut off (max_tokens). Increase max_tokens and retry.")
```

```
def _text(msg: anthropic.types.Message) -> str:
    return "".join(b.text for b in msg.content if b.type == "text")


def ask(system: str, user: str, *, max_tokens: int = 16000, effort: str = "medium",
        echo: bool = False) -> str:
    """Free-text generation. Streams so long outputs never hit HTTP timeouts."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
    ) as stream:
        if echo:
            for chunk in stream.text_stream:
                print(chunk, end="", flush=True)
            print()
        msg = stream.get_final_message()
    _check(msg)
    return _text(msg)


def ask_structured(system: str, user: str, model_cls: Type[T], *, max_tokens: int = 16000,
                   effort: str = "medium") -> T:
    """Structured generation: the SDK converts the Pydantic model to a JSON schema, the API
    constrains the output to it, and `parsed_output` is a validated instance of `model_cls`."""
    with client().messages.stream(
        model=MODEL,
        max_tokens=max_tokens,
        system=system,
        messages=[{"role": "user", "content": user}],
        thinking={"type": "adaptive"},
        output_config={"effort": effort},
        output_format=model_cls,
    ) as stream:
        msg = stream.get_final_message()
    _check(msg)
    if msg.parsed_output is None:  # should not happen with a constrained schema; be defensive
        return model_cls.model_validate_json(_text(msg))
    return msg.parsed_output
```

20.2.3 main.py

```
"""
AI PhD Supervisor Agent — "Virtual Research Mentor"
==================================================
A long-running mentor with file-based memory per student.

Usage
    python main.py init     --student "Ali" --topic "Multimodal AI for drought resilience in wheat" --years 3
    python main.py weekly   --student "Ali" --update "Finished RGB pipeline, accuracy 87%, hyperspectral data delayed"
    python main.py weekly   --student "Ali" --update-file this_week.md
    python main.py critique --student "Ali"
```

```
    python main.py present  --student "Ali" --type slides|poster|defence
    python main.py status   --student "Ali"

Memory: memory/<student>/ (profile.json, roadmap.json, progress_log.jsonl)   Reports: outputs/<student>/

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import sys
from datetime import date
from pathlib import Path

import typer
from rich.console import Console
from rich.panel import Panel
from rich.table import Table
from slugify import slugify

sys.path.insert(0, str(Path(__file__).parent))
if hasattr(sys.stdout, "reconfigure"):  # Windows consoles default to cp1252; we print → ✓ etc.
    sys.stdout.reconfigure(encoding="utf-8")
from agents import (CriticalReviewerAgent, PresentationAgent, ResearchPlanningAgent,  # noqa: E402
                    WeeklyProgressAgent)
from memory import StudentMemory  # noqa: E402

app = typer.Typer(add_completion=False)
console = Console()
OUT_DIR = Path(__file__).parent / "outputs"

BANNER = """[bold cyan]AI PhD Supervisor Agent[/] — Virtual Research Mentor
[dim]Codanics · Python ka Chilla: Build AI Agents · https://codanics.com/courses/python-ka-chilla-build-ai-agents/
YouTube @codanics · WhatsApp https://whatsapp.com/channel/0029Va7nRDq3QxRzGqaQvS3r[/]"""


def out_path(student: str, name: str) -> Path:
    p = OUT_DIR / slugify(student)
    p.mkdir(parents=True, exist_ok=True)
    return p / name


def require(mem: StudentMemory) -> None:
    if not mem.exists():
        console.print(f"[red]No memory for '{mem.student}'. Run: main.py init --student \"{mem.student}\" --topic ...[/]")
        raise typer.Exit(1)


@app.command()
def init(student: str = typer.Option(...), topic: str = typer.Option(...),
         years: int = typer.Option(3), background: str = typer.Option("", help="Degree, skills, prior work")):
    """Create the student's memory and a milestone-based roadmap."""
    console.print(Panel(BANNER, expand=False))
    mem = StudentMemory(student)
    roadmap = ResearchPlanningAgent().run(student, topic, years, background)
    mem.save_profile(name=student, topic=topic, years=years, background=background)
    mem.save_roadmap(roadmap.model_dump_json(indent=2))
    t = Table(title=roadmap.programme_title)
    t.add_column("sem"); t.add_column("milestone"); t.add_column("deliverables", max_width=70)
```

```
    for m in roadmap.milestones:
        t.add_row(str(m.semester), m.title, "; ".join(m.deliverables))
    console.print(t)
    md = [f"# PhD Roadmap — {student}\n", f"**Topic:** {topic}  \n**Length:** {years} years\n", "## Aims"]
    md += [f"- {a}" for a in roadmap.aims]
    for m in roadmap.milestones:
        md += [f"\n## Semester {m.semester}: {m.title}", *[f"- {d}" for d in m.deliverables],
               f"\n*Success criteria:* {m.success_criteria}  \n*Risks:* {', '.join(m.risks)}"]
    md += ["\n## Publications plan", *[f"- {p}" for p in roadmap.publications_plan],
           "\n## Skills to learn", *[f"- {s}" for s in roadmap.skills_to_learn],
           "\n## Weekly routine", *[f"- {w}" for w in roadmap.weekly_routine],
           "\n---\n*Codanics — Python ka Chilla: Build AI Agents · "
           "https://codanics.com/courses/python-ka-chilla-build-ai-agents/*"]
    out_path(student, "roadmap.md").write_text("\n".join(md), encoding="utf-8")
    console.print(Panel(f"Memory created in {mem.dir}\nRoadmap saved to outputs/{slugify(student)}/roadmap.md",
↪   style="green"))


@app.command()
def weekly(student: str = typer.Option(...), update: str = typer.Option(None),
           update_file: Path = typer.Option(None)):
    """Submit this week's progress update and get a structured evaluation."""
    console.print(Panel(BANNER, expand=False))
    mem = StudentMemory(student); require(mem)
    text = update or (update_file.read_text(encoding="utf-8") if update_file else None)
    if not text:
        raise typer.BadParameter("Provide --update or --update-file")
    ev = WeeklyProgressAgent().run(mem.context_summary(), text)
    mem.append_log({"update": text, "evaluation": ev.model_dump()})
    week = mem.week_number() - 1
    colour = {"on track": "green", "at risk": "yellow", "behind": "red"}.get(ev.status, "white")
    console.print(f"[bold]Week {week}[/] — progress [{colour}]{ev.progress_percent}% ({ev.status})[/]")
    console.print("[bold]Achievements:[/] " + "; ".join(ev.achievements))
    console.print("[bold]Problems:[/]")
    for p in ev.problems:
        console.print(f"  - {p}")
    console.print("[bold]Next week:[/]")
    for g in ev.next_week_goals:
        console.print(f"  - {g}")
    console.print(f"[bold]Supervisor should know:[/] {ev.supervisor_should_know}")
    console.print(f"[italic]{ev.encouragement}[/]")
    report = (f"# Weekly report — {student} — week {week} ({date.today().isoformat()})\n\n**Update:** {text}\n\n"
              f"**Progress:** {ev.progress_percent}% — {ev.status}\n\n## Achievements\n" +
              "\n".join(f"- {a}" for a in ev.achievements) + "\n\n## Problems\n" +
              "\n".join(f"- {p}" for p in ev.problems) + "\n\n## Goals for next week\n" +
              "\n".join(f"- {g}" for g in ev.next_week_goals) +
              f"\n\n**For the supervisor:** {ev.supervisor_should_know}\n\n*{ev.encouragement}*\n\n---\n"
              "*Codanics — Python ka Chilla: Build AI Agents ·
               ↪  https://codanics.com/courses/python-ka-chilla-build-ai-agents/*")
    out_path(student, f"week_{week:02d}.md").write_text(report, encoding="utf-8")


@app.command()
def critique(student: str = typer.Option(...), echo: bool = typer.Option(True)):
    """Challenge assumptions, methodology and novelty of the current work."""
    console.print(Panel(BANNER, expand=False))
    mem = StudentMemory(student); require(mem)
    text = CriticalReviewerAgent().run(mem.context_summary(last_n=8), echo=echo)
    p = out_path(student, f"critique_week_{mem.week_number() - 1:02d}.md")
```

```
    p.write_text(text, encoding="utf-8")
    console.print(Panel(f"saved {p}", style="green"))


@app.command()
def present(student: str = typer.Option(...),
            type: str = typer.Option("slides", help="slides | poster | defence"),
            echo: bool = typer.Option(True)):
    """Generate conference slides, a poster, or defence preparation."""
    console.print(Panel(BANNER, expand=False))
    mem = StudentMemory(student); require(mem)
    if type not in ("slides", "poster", "defence"):
        raise typer.BadParameter("--type must be slides, poster or defence")
    text = PresentationAgent().run(mem.context_summary(last_n=8), type, echo=echo)
    p = out_path(student, f"{type}.md")
    p.write_text(text, encoding="utf-8")
    console.print(Panel(f"saved {p}", style="green"))


@app.command()
def status(student: str = typer.Option(...)):
    """Show the memory summary (what the agent sees)."""
    mem = StudentMemory(student); require(mem)
    console.print(mem.context_summary())


if __name__ == "__main__":
    app()
```

20.2.4 `memory.py`

```
"""
memory.py — simple, transparent file-based memory for one PhD student.

memory/<student-slug>/
    profile.json        who the student is, topic, start date
    roadmap.json        the 3-year plan (milestones)
    progress_log.jsonl  one JSON line per weekly update + evaluation (append-only audit trail)

The agent never receives the whole history — it receives a compact `context_summary()`:
profile + roadmap + the last N updates.  This keeps prompts small and focused.

Codanics — Python ka Chilla: Build AI Agents
https://codanics.com/courses/python-ka-chilla-build-ai-agents/
"""
from __future__ import annotations

import json
from datetime import date
from pathlib import Path

from slugify import slugify

MEMORY_ROOT = Path(__file__).parent / "memory"
```

```python
class StudentMemory:
    def __init__(self, student: str):
        self.student = student
        self.dir = MEMORY_ROOT / slugify(student)
        self.dir.mkdir(parents=True, exist_ok=True)
        self.profile_path = self.dir / "profile.json"
        self.roadmap_path = self.dir / "roadmap.json"
        self.log_path = self.dir / "progress_log.jsonl"

    # ---- profile / roadmap ----------------------------------------------------
    def exists(self) -> bool:
        return self.profile_path.exists() and self.roadmap_path.exists()

    def save_profile(self, **fields) -> None:
        fields.setdefault("start_date", date.today().isoformat())
        self.profile_path.write_text(json.dumps(fields, indent=2), encoding="utf-8")

    def profile(self) -> dict:
        return json.loads(self.profile_path.read_text(encoding="utf-8"))

    def save_roadmap(self, roadmap_json: str) -> None:
        self.roadmap_path.write_text(roadmap_json, encoding="utf-8")

    def roadmap(self) -> dict:
        return json.loads(self.roadmap_path.read_text(encoding="utf-8"))

    # ---- progress log ---------------------------------------------------------
    def append_log(self, entry: dict) -> None:
        entry.setdefault("date", date.today().isoformat())
        entry["week"] = self.week_number()
        with self.log_path.open("a", encoding="utf-8") as f:
            f.write(json.dumps(entry, ensure_ascii=False) + "\n")

    def log(self) -> list[dict]:
        if not self.log_path.exists():
            return []
        return [json.loads(l) for l in self.log_path.read_text(encoding="utf-8").splitlines() if l.strip()]

    def week_number(self) -> int:
        return len(self.log()) + 1

    # ---- what the LLM sees ----------------------------------------------------
    def context_summary(self, last_n: int = 4) -> str:
        p, r = self.profile(), self.roadmap()
        recent = self.log()[-last_n:]
        lines = [f"STUDENT: {p.get('name')} | topic: {p.get('topic')} | start: {p.get('start_date')} | "
                 f"programme length: {p.get('years')} years | current week: {self.week_number()}",
                 "ROADMAP:", json.dumps(r, indent=1)[:6000],
                 f"RECENT PROGRESS (last {len(recent)} of {len(self.log())} updates):"]
        for e in recent:
            ev = e.get("evaluation", {})
            lines.append(f"- week {e.get('week')} ({e.get('date')}): {e.get('update')[:400]}\n"
                         f"  progress={ev.get('progress_percent')} status={ev.get('status')} "
                         f"problems={ev.get('problems')} next={ev.get('next_week_goals')}")
        return "\n".join(lines)
```