AI tools and APIs

Models, agents, reproducible workflows, and programmatic access

Day 4 of 4 · 12 August 2026

Fatih Kansoy
Worcester College, University of Oxford

Outline

1. Practical sessions
2. Model inputs, context, and routing
3. Agents and task execution
4. Editors and reproducible documents
5. Hosted, open, and local models
6. Development workflow and voice input
7. APIs and programmatic automation

Practical 01: Make safe API calls and compare models

public API $\longrightarrow$ protected model key $\longrightarrow$ measured comparison

You will

- call a public weather API and inspect its JSON response;
- load model keys from a `.env` file;
- send one model request and calculate its token cost.

Then compare

- a low-cost and a mid-tier model on labelled headlines;
- accuracy, response time, and cost;
- several model providers through OpenRouter.

Saved result: model answers, prompts, model names, date, and comparison metadata.

Practical 02: Label central-bank speeches at scale

download $\longrightarrow$ filter $\longrightarrow$ measure climate mentions $\longrightarrow$ sample $\longrightarrow$ label

You will

- download and cache the complete BIS speech archive;
- filter by date, year, month, and country or author text;
- chart explicit climate references over time;
- label a reproducible sample for topic, inflation stance, and summary;
- repeat the task with two model tiers and compare their outputs.

Research decisions
Define the climate terms and labels, record every filter and truncation rule, inspect errors, and use measured token counts to price the selected corpus.

Saved result: climate series, labels, provenance, and figures.

Practical 03: Build checked country briefings

World Bank API ⟶ writer model ⟶ checker model ⟶ human review

Build the briefings

- download inflation, growth, and unemployment data;
- prepare a compact table for each country;
- instruct the writer to use only that table.

Verify the briefings

- use a model from another provider to inspect every claim;
- return a structured pass or fail result;
- review failed cases and save one editable briefing document.

Saved result: a chart, verification results, and `briefings.md`.

Practical 04: Compare two agent teams

The same written mission is given to **Claude Code** and to **OpenCode with DeepSeek**. Each system must build the complete pipeline.

$$\text{scrape} \longrightarrow \text{analyse} \longrightarrow \text{visualise} \longrightarrow \text{report}$$

The agent roles
Scraper, analyst, visualiser, and reporter. The orchestrator verifies each stage before the next role begins.

Your comparison
Evaluate both runs against the same acceptance checklist, then compare data, findings, figures, report quality, cost, and completion time.

Deliverables: 1,000 books, summary statistics, six figures, runnable code, and a short analytical report.

Part I

Model inputs, context, and routing

How model requests are defined and assigned.

A model request: input, reasoning, output

Input

Instructions, files, images, tool results

→

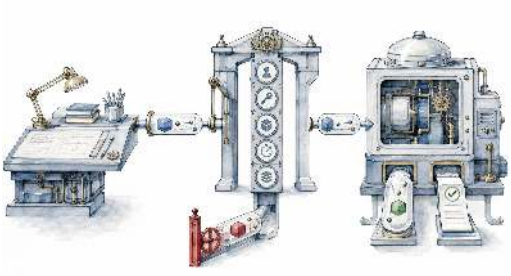
Reasoning

Model computation used to form the answer

→

Output

Text, structured data, or a tool call



Reasoning effort controls how much work the model spends on this request. More effort can help difficult tasks, but takes longer.

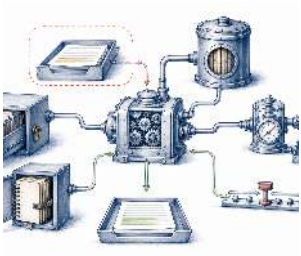
Context is what the model can see now

1.05M tokens

≈ 790,000 English words

≈ 8–10 ordinary books

The prompt, files, conversation, tool results, and answer must fit inside this working space.



Knowledge cutoff

16 February 2026

Use search, files, or APIs for newer facts.

Model tiers differ in cost and capability

Luna

`gpt-5.6-luna`

Routine, high-volume work

1× token price

Terra

`gpt-5.6-terra`

Balanced analysis and creation

2.5× token price

Sol

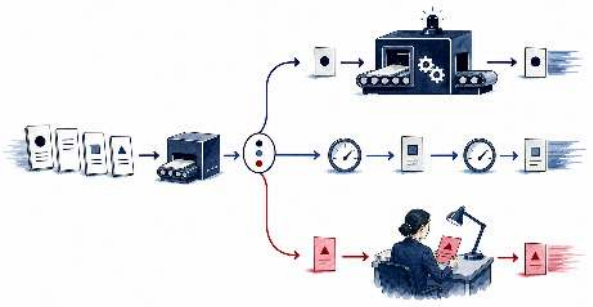
`gpt-5.6-sol`

Difficult reasoning and coding

5× token price

All three share the same context window, output limit, and knowledge cutoff.

Assign model tiers by task difficulty



Luna
Extract, classify, summarize

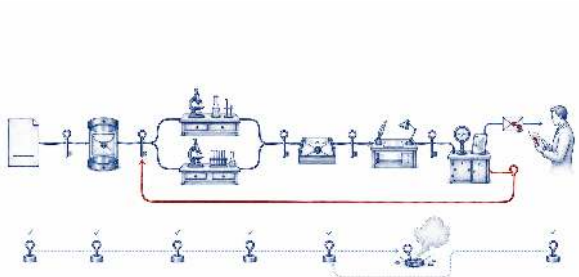
Terra
Analyse, code, design charts

Sol
Plan and review difficult work

Escalate only when the cheaper route fails a test or the decision is important.

Model orchestration separates planning, production, and review

Sol plans → **Luna** processes → **Terra** builds → **Sol** reviews → **Human** approves



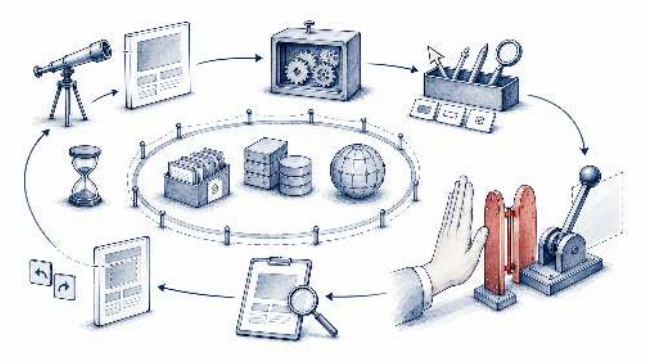
If review fails, return the work to Terra. Keep human approval for consequential actions.

Part II

Agents and task execution

How tools, state, and permissions support multi-step work.

Agents run a repeated action loop



Observe
See the current state.

Decide
Choose the next step.

Act
Use an allowed tool.

Check
Stop, continue, or ask.

Benefit: one request can become a multi-step task across applications.

OpenClaw and Hermes provide agent infrastructure

OpenClaw

Connects your chat channels to tools, skills, files, terminals, and browsers.

Benefit: reach one assistant from the communication tools you use.

OpenClaw documentation

Hermes Agent

Adds persistent memory, reusable skills, scheduling, and delegation.

Benefit: repeated workflows retain context and procedures.

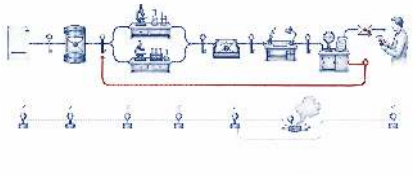
Hermes documentation

Both are open-source agent frameworks. The user supplies the model and defines the available tools and permissions.

Agent workflows require clear limits

Example: a morning research brief

Search approved sources $\longrightarrow$ summarize $\longrightarrow$ save notes $\longrightarrow$ draft the message



What you gain
Less app switching, scheduled work, reusable memory, and an action record.

Approval boundary
Allow only needed accounts and tools. Sending, deleting, purchasing, publishing, or changing an external system requires approval.

Part III

Editors and reproducible documents

How text-based source supports review, automation, and reuse.

Editors organise text-based projects

Files	Keep text, figures, data, code, and references together.
Editor	Search, replace, navigate, and make precise changes.
Terminal	Build, test, convert, and publish without changing tools.
Git	Review a diff and recover an earlier version.

An IDE adds language tools, debugging, extensions, and AI assistance around the editor. The source files remain the durable record.

VS Code editor overview

Plain text supports review and automation

Word processor file		Text source
Content and formatting are stored inside a complex document.	$\longrightarrow$	Content and structure are readable lines in a file.
Excellent for familiar visual editing, comments, and tracked changes.		Easy to search, compare, generate, validate, and edit with an agent.

A visible source is easier to review and automate. The final output can still be PDF, HTML, Word, or slides.

Reproducible documents require source files and a build command

source text + data + code + references $\longrightarrow$ build $\longrightarrow$ document or presentation

What improves

- the same inputs can regenerate the output;
- figures and numbers can update from code;
- Git records who changed each line;
- automated checks can catch missing files or failed builds.

What must be recorded

- software and package versions;
- data and figure locations;
- fonts, templates, and build settings;
- the command that produces the final artifact.

Text-based publishing systems serve different outputs

Markdown Simple, readable notes and documentation. Strong default when structure matters more than exact page design.

LaTeX / Beamer Precise equations, citations, technical documents, and lecture slides. Powerful, but the learning curve and build errors are real.

Quarto Markdown plus code, citations, figures, and cross-references. One source can produce HTML, PDF, Word, websites, and presentations.

Typst A modern markup and typesetting system with fast compilation and strong PDF output. Often easier to learn than LaTeX.

Use the simplest source format that produces the required output. Exact typesetting control adds value when layout, equations, or citations demand it.

Quarto guide **Typst documentation**

Part IV

Hosted, open, and local models

How access routes change cost, control, and operations.

Other hosted model families

They add strong alternatives for:

- low-cost hosted inference;
- downloadable weights and local deployment;
- Chinese and multilingual work;
- coding, reasoning, and multimodal applications.

Representative families:

DeepSeek **Qwen**

Kimi **GLM**

Each family contains several model versions. Record the exact version, provider, licence, data policy, and local evaluation.

Compare models on the same task and operating constraints.

DeepSeek increased price and openness competition

Why it mattered

- strong models arrived with unusually low hosted prices;
- major releases published weights and technical detail;
- researchers and firms could study, adapt, and host released models, subject to each licence.

The important change was competitive pressure: high capability no longer implied one closed provider and a high API price.

DeepSeek V4 Flash 0423

$0.14 input
$0.28 output

per 1M uncached tokens

Official API price, checked 12 August 2026

OpenRouter lists 0423 and 0731 as separate routes. Record the full model ID; prices and speed change.

Current DeepSeek pricing

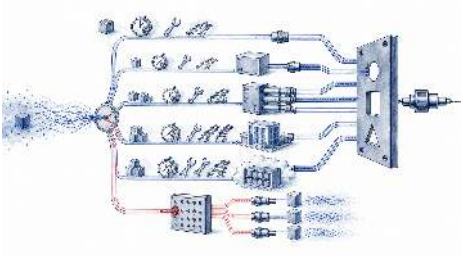
OpenRouter routes one API to multiple providers

What it does

One interface reaches many models and inference providers.

Why use it

- compare models without rewriting the application;
- route by price, latency, throughput, or provider;
- use fallbacks and consolidated billing.



The upstream provider still controls retention, region, quota, and acceptable-use rules. Log the resolved model and provider.

OpenRouter provider routing

Traycer coordinates coding agents through a reviewed plan

intent $\longrightarrow$ specification $\longrightarrow$ plan $\longrightarrow$ agent handoff $\longrightarrow$ verification

What it does
Traycer works inside supported editors and coordinates coding agents around a reviewed plan. It can separate a large change into phases and verify the result against the plan.

What it adds
Less prompt drift, clearer handoffs, visible acceptance criteria, and a shared record when several agents or people work on the same codebase.

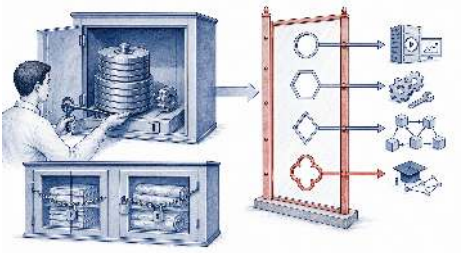
Traycer supplies orchestration and verification. A separate model or coding agent performs the implementation.
Traycer documentation

Open weights provide downloadable model parameters

With the files and suitable software, you may be able to:

- run the model on your own hardware;
- quantize it to reduce memory use;
- fine-tune or adapt it;
- deploy it through your own API.

Only if the licence permits the intended use.



An open-weight release may exclude the training data, training code, and full recipe. The licence defines allowed use, and hosting still has a cost.

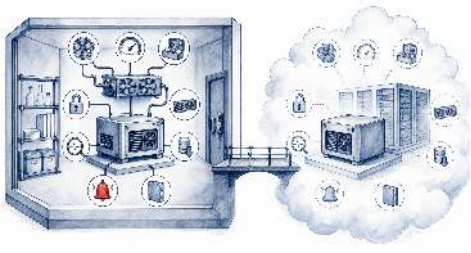
Ollama and LM Studio run models locally

Ollama

Command-line-first runtime with a local API. Useful for scripts, editors, coding agents, and repeatable development environments.

LM Studio

Desktop interface for finding, testing, and serving local models. Useful when you want a visual model manager and chat interface.



Both can expose local inference to other applications. They provide runtimes and interfaces; the user selects the model separately.

Local runtimes expose models through an API

download weights $\longrightarrow$ load runtime $\longrightarrow$ local API $\longrightarrow$ application

Ollama
Default local API: `localhost:11434`. A script can change the model name without changing the whole workflow.

LM Studio
Start a local server from the interface or CLI. OpenAI-compatible endpoints let many existing applications connect with a base-URL change.

Select a quantized model that fits memory. Measure task quality, tokens per second, time to first token, and memory use.

Local deployment changes control, cost, and operations

Choose local when you need

- offline work or low network dependence;
- control of the model and data path;
- predictable device latency;
- a narrow repeated workload at sufficient volume.

Account for

- lower quality from smaller or quantized models;
- RAM/VRAM, speed, energy, and storage limits;
- access control, logs, patches, monitoring, and backups;
- licence and model-version management.

Local prompts stay local only when the selected model and features are local. Model search, downloads, updates, web search, and cloud models can still use the network.

Small models suit narrow, repeated business tasks

A small language model has no universal size boundary. It is small enough to run economically on the available device or server for a bounded task.

Strong business fits include:

- classifying messages or documents;
- extracting known fields into a schema;
- routing requests to the correct workflow;
- drafting from approved templates;
- on-device or offline assistance.

The advantage is lower latency, predictable unit cost, and deployment control. The cost is less broad knowledge and less reserve for ambiguous cases.

A cascade sends difficult cases to a stronger route

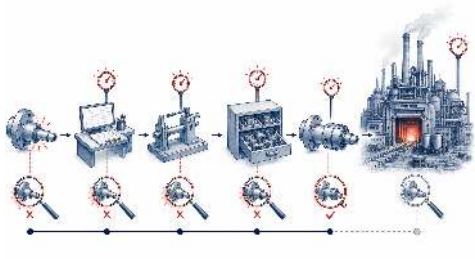
small model

↓

deterministic check

↓

accept or escalate



Escalate ambiguous cases to a stronger model and consequential cases to a person. Measure cost per accepted result, not the small model's token price alone.

Part V

Development workflow and voice input

How browser tools, version control, deployment, and dictation fit the workflow.

Replit provides a browser-based development environment

Replit combines a browser editor, runtime, preview, collaboration, AI agent, and deployment path.

Why it helps

- almost no local setup;
- easy sharing for teaching and prototypes;
- code, running output, and deployment stay close together.

Production requirements: version control, secrets, tests, runtime limits, cost, export path, and platform dependence.

Replit documentation

Git records changes to a project

Git stores a sequence of named project versions called **commits**.

edit $\longrightarrow$ inspect the diff $\longrightarrow$ commit $\longrightarrow$ continue

History
Compare versions, identify when a change was introduced, and restore an earlier state.

Branches
Develop an alternative without changing the stable branch, then merge the reviewed result.

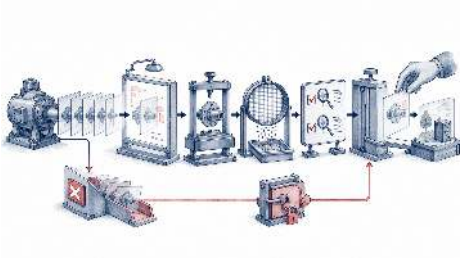
Git works well with source text because each changed line is visible. Maintain separate backups for data, credentials, and generated files.

GitHub supports collaboration, automation, and deployment

Repository hosting
Store a shared Git repository with access controls and an issue record.

Pull requests
Discuss a proposed change, review the diff, run tests, and approve the merge.

GitHub Actions
Build documents, test code, publish a site, or deploy an application when the repository changes.



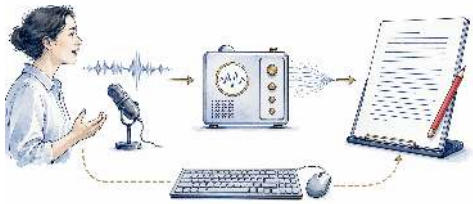
Protected environments can restrict deployment branches, secrets, and approvals. GitHub supports the workflow; the project still defines the tests, review rules, and release process.

Voice dictation can speed up first drafts

Speaking can create a first draft faster than typing, especially for:

- prompts, emails, and notes;
- outlining while reading or walking;
- accessibility and reduced keyboard load;
- capturing ideas quickly for later editing.

Wispr Flow provides dictation across applications. System dictation is a free starting point.



Treat the transcript as a draft: structure, verify, and edit it.

Speech transcripts require verification

speak $\longrightarrow$ text $\longrightarrow$ review $\longrightarrow$ use

Review names, numbers, technical terms, quotations, and commands. Sending, deleting, purchasing, or publishing requires confirmation.

For products and agents, a speech-to-text API such as **ElevenLabs Scribe** can provide batch or real-time transcription.

The data policy must state where audio and transcripts are processed and retained.

Part VI

APIs and programmatic automation

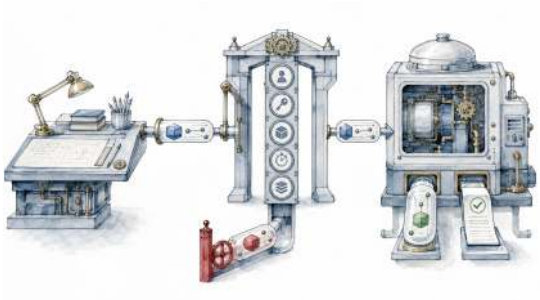
How programs call services safely and repeatedly.

API contracts define program-to-program access

An **API** is a published agreement between programs:

- where a request is sent;
- which input fields are accepted;
- what the response contains;
- how access, limits, and errors work.

The service can change internally while the agreement remains stable.



A documented call can be tested, repeated, and placed inside a larger workflow.

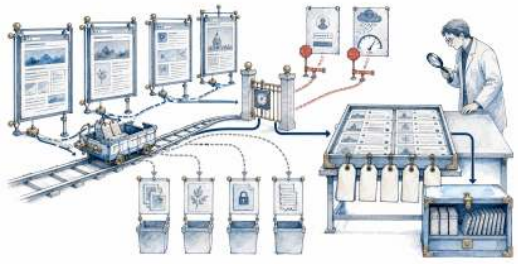
APIs and browser automation provide different access

API

The service publishes fields, response formats, authentication, and limits. This is usually the more stable path for automation.

Browser or scraping

Software reads or controls a human-facing page. Use it when the required information or action has no suitable API.



Browser access is more fragile. Respect access controls, terms, rate limits, and personal-data rules.

Programs call AI models through APIs

input $\longrightarrow$ API request $\longrightarrow$ model $\longrightarrow$ structured response

The program supplies

- model identifier;
- instructions and content;
- output format and tool definitions;
- limits such as maximum output.

The program receives

- generated text or structured data;
- a proposed tool call;
- usage and status information;
- an error when the request cannot run.

Changing the model identifier is easy. The new route must still be retested for quality, format, latency, and cost.

APIs support repeated analysis at scale

Example: central-bank speeches

Ask the same documented question of every permitted speech, save the structured responses, and inspect failures.

$19{,}000 \times 3{,}000 \approx 57$ million input tokens

Estimated model cost

input tokens $\times$ input price + output tokens $\times$ output price

Scale requires more than a loop: validation, rate-limit handling, retries, version records, and a review sample.

Store API keys outside source code

The key identifies the account and can spend its quota.

Keep it out of

- source code and notebooks;
- Git repositories;
- slides, email, and screenshots;
- browser code sent to users.

Keep it in

- an environment variable for local work;
- a managed secret store in production;
- a restricted project with usage limits;
- a rotation process if exposure is suspected.

Practical 01: store one key safely, send one request, and inspect its response. Automation comes later.