Chain

A chain is a pipeline that turns an **input** (e.g., a user query) into an **output** by passing it through steps (prompts, LLM calls, parsers, small functions). LangChain's Runnable system lets you compose these steps into three common topologies:

1. **Sequential** (do A → then B → then C)
2. **Parallel** (do A, B, C at the same time, then combine)
3. **Branching** (choose one path among many based on a rule)

---

1) Sequential Chain

What it is

A straight pipeline: the output of one step becomes the input to the next. Think "assembly line."

How data flows

input → Step 1 → Step 2 → Step 3 → output

Each step receives the previous step's result. If a step needs additional fields, you enrich the data as it moves along (e.g., attach user_id or timestamp).

When to use

- Multi-pass prompting (draft → critique → improve → finalize).
- ETL-like flows (extract entities → normalize → format JSON → validate).
- "Reason then act" patterns (plan → execute tool → summarize results).

Example (no code)

- **Use case:** Create a market brief.
    1. **Step 1:** "Write a detailed report on the product."
    2. **Step 2:** "From that report, extract 5 key bullets."
    3. **Step 3:** "Turn those bullets into an executive summary paragraph."
       Output: A polished summary based on earlier steps.

Strengths & pitfalls

- **Strengths:** Simple, predictable; easy to debug step-by-step.
- **Pitfalls:** Slow if each step waits for the previous; fragile if an early step returns malformed data (so add validation/parsers).

---

2) Parallel Chain

What it is

Runs multiple sub-chains **at the same time** on the **same input**, then collects their outputs into one result (fan-out → fan-in).

How data flows

input → [Branch A | Branch B | Branch C] → merged output (a dict-like bundle)

Each branch gets the same input and does its own job. After execution, their outputs are merged (e.g., {summary: …, keywords: …, sentiment: …}).

When to use

- Enrich one input with several independent views: summarize, extract entities, score sentiment, generate tags.
- Speed up work by parallelizing independent tasks.
- Multi-model agreement (ask different models/temperatures in parallel, compare/merge results).

Example (no code)

- **Use case:** Analyze a product review.
    - **Branch A:** Summarize the review in 2–3 sentences.
    - **Branch B:** Extract 5 keywords.
    - **Branch C:** Classify sentiment (positive/neutral/negative).
      Merged output: {summary, keywords, sentiment} returned together.

Strengths & pitfalls

- **Strengths:** Faster total time for independent tasks; modular outputs.
- **Pitfalls:** You must reconcile conflicts (e.g., two branches disagree); outputs must be named clearly to avoid collisions.

---

3) Branch Chain (Routing)

What it is

Chooses **one** of several possible paths based on a routing rule or predicate. Think of it as a switch: "If condition X → go down path A; else if Y → path B; else → default."

How data flows

input → router (predicate checks) → chosen branch → output

Only one branch runs per input (unless you explicitly design it otherwise).

When to use

- Tool selection: "If the user asks for weather → call weather tool; if finance → call stock chain."
- Tiered prompting: "Short inputs use quick prompt; long inputs use summarization pipeline."
- Policy/guardrails: "If content unsafe → safe response chain; else → normal chain."

Example (no code)

- **Use case:** Customer question triage.
    - **Router rule:**
        - If the text contains shipping terms → route to **Shipping FAQ chain**.
        - If it contains refund terms → route to **Refund policy chain**.
        - Otherwise → route to **General support chain**.
          Output: Best-matched answer using only the selected branch.

Strengths & pitfalls

- **Strengths:** Efficiency (only one branch runs); clarity (each branch specialized).
- **Pitfalls:** Routing errors can misclassify inputs; define good predicates and keep a safe default branch.

Core building blocks

RunnableParallel

- **What it does:** Executes multiple named sub-runnables concurrently on the same input and returns a **single combined mapping** of their outputs.
- **Mental model:** A “fan-out/fan-in” splitter/merger.
- **Good for:** Multi-extraction, multi-summary, ensemble prompts, multi-index retrieval.
- **Gotchas:**
    - Ensure each sub-output has a unique key.
    - Plan for post-merge normalization: different sub-chains may return different shapes.

RunnableBranch

- **What it does:** Evaluates predicates in order and runs the **first** matching branch; optionally a default branch if none match.
- **Mental model:** if / elif / else for chains.
- **Good for:** Dynamic tool choice, policy gating, content-type routing, “cold path vs hot path” optimization.
- **Gotchas:**
    - Predicates should be fast and deterministic where possible.
    - Always include a safe default branch to prevent dead ends.

RunnableLambda

- **What it does:** Wraps a **small Python function** as a runnable step. It’s the glue for transforming data between steps: rename fields, compute new values, select a property from a dict, filter/flatten lists, etc.
- **Mental model:** A lightweight adapter/transformer node between bigger steps.
- **Good for:**
    - Mapping {report: "..."} into {text: "..."} for the next prompt.
    - Casting types (string → float), trimming text, joining arrays, building tool arguments.

- **Gotchas:**
    - Keep side effects out (treat it as pure if you want determinism/caching).
    - Make sure the function returns exactly the shape the next step expects.

Putting it all together (pattern cookbook)

1. **Sequential with validation:**
    - Step A (LLM) → Step B (Structured/Pydantic parser) → Step C (LLM) → Step D (final formatter).
    - Use Lambda steps to rename/massage fields between B and C.
2. **Parallel enrichment then synthesize:**
    - Fan-out: summary, keywords, sentiment, entities in parallel.
    - Fan-in: combine into a single object; optionally run a final "synthesis" prompt that takes all fields and writes a single answer.
3. **Branch by intent then unify shape:**
    - Router picks "Weather chain" vs "Finance chain" vs "General chain."
    - Each branch returns a common shape (e.g., {answer, sources}) so downstream consumers don't care which path ran.