# Atomic Task Decomposition – Separated Workflows

This document follows the style of the project’s atomic pipeline breakdown. Each workflow identified in the repository is decomposed into atomic tasks. Tasks are presented as YAML‑friendly identifiers (atom\_xxx) with descriptive names and optional comments. Where relevant, citations reference the source lines that describe the task.

## Multi‑CLI Orchestrator (MO)

The TaskOrchestrator class orchestrates code edits by analysing task complexity, selecting tools and executing commands. Its atoms are grouped by function.

# Phase MO0 – Context and Analysis  
mo\_atom\_000: create\_context\_object # initialize context with ID and metadata  
mo\_atom\_001: record\_start\_time\_and\_prompt # capture start time and user prompt  
mo\_atom\_002: count\_files\_and\_lines # compute number of files and total lines  
mo\_atom\_003: scan\_for\_keywords # search prompts/files for complexity keywords  
mo\_atom\_004: compute\_complexity\_score # derive complexity level (simple/moderate/complex)  
  
# Phase MO1 – Planning  
mo\_atom\_005: select\_toolchain\_based\_on\_complexity # choose Aider/Copilot/Claude or fallback  
mo\_atom\_006: check\_hard\_constraints # verify git clean and path allowlist (not explicitly shown but implied)  
mo\_atom\_007: assemble\_execution\_plan # build ordered list of tools and steps  
  
# Phase MO2 – Execution  
mo\_atom\_008: invoke\_cli\_tool # run tool command for current step  
mo\_atom\_009: capture\_tool\_output # store stdout/stderr for parsing  
mo\_atom\_010: detect\_file\_changes # check which files were modified  
mo\_atom\_011: parse\_delegation\_suggestions # inspect output for recommended delegation  
mo\_atom\_012: update\_plan\_with\_delegations # adjust plan if another tool is suggested  
mo\_atom\_013: iterate\_over\_plan # loop through remaining steps  
  
# Phase MO3 – Summary  
mo\_atom\_014: compile\_results\_summary # summarise tools used and reasoning  
mo\_atom\_015: emit\_final\_output # return summary to caller

## Free‑Tier Orchestrator (FT)

The free\_tier\_orchestrator.ps1 script manages lanes and token quotas. Atoms are grouped by command.

# Command: init  
ft\_atom\_000: create\_ai\_directory\_and\_gitignore # ensure .ai directory exists  
ft\_atom\_001: initialize\_quota\_tracker # create quota‑tracker.json  
ft\_atom\_002: select\_best\_service\_by\_quota # choose local/HF/Claude based on token availability  
ft\_atom\_003: pull\_local\_models\_with\_ollama # download models like codellama/phi  
ft\_atom\_004: verify\_model\_runtime # test that models run successfully  
  
# Command: init‑lane  
ft\_atom\_005: create\_git\_worktree\_for\_lane # git worktree add for new lane  
ft\_atom\_006: copy\_configuration\_to\_lane # clone .ai and tool configs into lane  
ft\_atom\_007: register\_lane\_watchers # configure watchers and deadlines  
  
# Command: start‑lane  
ft\_atom\_008: verify\_lane\_exists # check branch/worktree presence  
ft\_atom\_009: display\_vscode\_command # show code --reuse-window command  
ft\_atom\_010: list\_available\_tools\_and\_deadlines # print allowed tools and time limits  
  
# Command: submit‑lane  
ft\_atom\_011: validate\_changed\_files # ensure only allowed files changed  
ft\_atom\_012: run\_precommit\_checks # ruff/mypy/pytest pre‑commit scripts  
ft\_atom\_013: commit\_and\_push\_lane # commit with message and push to remote  
ft\_atom\_014: notify\_watchers\_of\_submission # send notifications upon submission  
  
# Command: integrate  
ft\_atom\_015: merge\_lane\_into\_integration # reset integration branch and merge lane  
ft\_atom\_016: run\_integration\_tests # execute project’s integration tests  
ft\_atom\_017: fast\_forward\_or\_rollback # fast‑forward on success or revert on failure  
  
# Command: status  
ft\_atom\_018: show\_token\_quota\_status # display spent and remaining tokens  
ft\_atom\_019: report\_lane\_statuses # list active lanes and integration status

## Granular Multi‑Phase Workflow (GF)

This document outlines a robust multi‑phase workflow for large tasks. The phases and tasks are captured below.

# Phase GF1 – System Preparation & Inventory  
gf\_atom\_000: create\_ai\_directory\_and\_worktrees # ensure .ai and multiple worktrees exist  
gf\_atom\_001: initialize\_quota\_tracker\_and\_tools # set up quota tracker and validate tools  
gf\_atom\_002: scan\_file\_inventory # list files and gather size/line metrics  
gf\_atom\_003: parse\_project\_documentation # read README and docs for requirements  
  
# Phase GF2 – Gap Analysis & Routing  
gf\_atom\_004: perform\_gap\_analysis # compare current code with requirements  
gf\_atom\_005: estimate\_task\_complexity # assign complexity categories and token counts  
gf\_atom\_006: route\_tasks\_via\_decision\_matrix # map tasks to simple/moderate/complex lanes  
gf\_atom\_007: request\_user\_approval\_for\_premium # prompt user for cost approvals  
  
# Phase GF3 – Execution  
gf\_atom\_008: init\_simple\_worktree\_and\_run\_gemini # create simple lane and run gemini CLI  
gf\_atom\_009: auto\_test\_and\_commit\_simple\_changes # run tests and auto‑commit successful changes  
gf\_atom\_010: init\_moderate\_worktree\_and\_run\_aider # create moderate lane and engage Aider  
gf\_atom\_011: run\_tests\_and\_commit\_moderate # execute tests and commit moderate lane  
gf\_atom\_012: init\_complex\_worktree\_and\_run\_claude # set up complex lane and call Claude Code  
gf\_atom\_013: implement\_complex\_features # multi‑step code generation (models, routes, tests)  
gf\_atom\_014: triage\_failing\_tests\_with\_vscode # start VS Code diagnostics and export report  
gf\_atom\_015: categorize\_and\_route\_issues # classify issues into auto‑fix, Aider or manual  
  
# Phase GF4 – Integration & Quality Assurance  
gf\_atom\_016: resolve\_dependencies\_and\_update # align dependencies across lanes  
gf\_atom\_017: run\_integration\_tests\_and\_fixtures # execute integration tests and fix issues  
gf\_atom\_018: perform\_security\_and\_quality\_scans # run bandit/safety/lint  
  
# Phase GF5 – Merge & Finalization  
gf\_atom\_019: sequential\_merge\_branches # merge lanes back sequentially  
gf\_atom\_020: conduct\_final\_validation # final smoke tests and manual checks

## Symbiotic CLI Workflows (SY)

These workflows describe how Claude Code and OpenAI Codex collaborate. Each workflow is treated separately.

# Workflow SY1 – Feature Implementation & Tests  
sy1\_atom\_000: capture\_feature\_request # parse high‑level feature description  
sy1\_atom\_001: run\_claude\_for\_planning\_and\_impl # Claude analyses code and implements changes  
sy1\_atom\_002: identify\_modified\_files # list files/functions changed by Claude  
sy1\_atom\_003: generate\_unit\_tests\_with\_codex # use Codex to write tests for each change  
sy1\_atom\_004: run\_tests\_and\_iterate # run tests; if failures, loop back to Claude  
sy1\_atom\_005: commit\_successful\_changes # commit once tests pass  
  
# Workflow SY2 – Refactoring & Documentation  
sy2\_atom\_000: refactor\_code\_with\_claude # perform structural refactoring  
sy2\_atom\_001: add\_docstrings\_and\_types\_with\_codex # Codex adds docstrings/type hints  
sy2\_atom\_002: run\_tests\_and\_commit # test and commit changes after docs  
  
# Workflow SY3 – Cross‑Language Tasks  
sy3\_atom\_000: define\_cross\_language\_goal # describe backend + frontend feature  
sy3\_atom\_001: implement\_backend\_with\_claude # Claude writes backend code  
sy3\_atom\_002: implement\_frontend\_with\_codex # Codex produces JS/TS snippet  
sy3\_atom\_003: integrate\_and\_test # combine both sides and test  
sy3\_atom\_004: commit\_final\_implementation # finalize after passing tests

## Routing Engine (RT)

The IntelligentRouter evaluates tasks to select the most appropriate tool. Its atoms follow the decision flow.

rt\_atom\_000: check\_git\_clean\_and\_paths # enforce hard constraints; else route to VS Code  
rt\_atom\_001: compute\_file\_and\_line\_counts # count files, lines and average sizes  
rt\_atom\_002: assess\_prompt\_complexity # measure complexity of the prompt  
rt\_atom\_003: calculate\_overall\_complexity\_score # compute weighted complexity  
rt\_atom\_004: evaluate\_tool\_candidates # estimate cost/duration for each tool  
rt\_atom\_005: select\_tool\_with\_confidence # choose the best tool and explain reasoning

## Predetermined Workflow – PY\_EDIT\_V2 (PW)

This deterministic workflow always follows the same path based on file counts and sizes.

# Phase PW0 – Preparation  
pw\_atom\_000: initialize\_workflow\_context # record start state[1]  
pw\_atom\_001: evaluate\_git\_and\_file\_constraints # check git clean and allowed paths[2]  
  
# Phase PW1 – Tool Selection  
pw\_atom\_002: decide\_between\_aider\_claude\_vscode # choose aider, claude or vscode based on size[3][4][5]  
  
# Phase PW2 – Execution  
pw\_atom\_003: run\_selected\_tool # execute predetermined CLI command[3][4]  
pw\_atom\_004: perform\_quality\_checks # run ruff, mypy and pytest[3][4]  
  
# Phase PW3 – Finalization  
pw\_atom\_005: commit\_with\_conventional\_message # commit changes with prefix[3][4]  
pw\_atom\_006: push\_changes\_to\_remote # push to remote repository[3][4]

## Workflow Enhancements (WH)

These atoms provide observability and logging features.

wh\_atom\_000: capture\_pre\_and\_post\_git\_snapshots # record branch, hash, commits and uncommitted files[6]  
wh\_atom\_001: generate\_run\_identifier # create run ID like yyyyMMdd-HHmmss-hex[7]  
wh\_atom\_002: display\_execution\_banner # show banner with workflow name and run ID[8]  
wh\_atom\_003: produce\_exit\_summary # report duration, steps, tokens and git changes[9]  
wh\_atom\_004: write\_manifest\_to\_artifacts # output manifest json with snapshots and stats[10]  
wh\_atom\_005: log\_activity\_events # write structured logs to logs/workflow\_execution.log[11]  
wh\_atom\_006: rotate\_logs\_as\_needed # perform log rotation beyond size limits[12]

## Simplified Workflow Mode (SW)

This mode converts declarative operations into a lightweight execution plan.

sw\_atom\_000: enable\_simplified\_mode # set simplified: true or provide operations list[13]  
sw\_atom\_001: map\_operations\_to\_roles # use RoleManager for 5-role mapping[13]  
sw\_atom\_002: select\_tool\_via\_simplified\_router # run SimplifiedRouter decision matrix[13]  
sw\_atom\_003: estimate\_tokens\_with\_costtracker # log token usage estimates before execution[13]  
sw\_atom\_004: execute\_or\_dry\_run\_operations # convert operations to steps and run via adapters[14]

## API Execution Workflows (AP)

Atoms here describe how to run workflows via the HTTP API.

ap\_atom\_000: post\_workflow\_execution\_request # send POST request with workflow\_file, files, lane and tokens[15]  
ap\_atom\_001: parse\_execution\_response # extract execution\_id, success, artifacts and metrics[16]  
ap\_atom\_002: execute\_dry\_run # call API with dry\_run: true for preview[17]  
ap\_atom\_003: pass\_custom\_parameters # include analysis\_depth, fix\_suggestions, output\_format etc.[18]  
ap\_atom\_004: handle\_large\_codebase\_parameters # set max\_tokens and exclude paths accordingly[19]  
ap\_atom\_005: interpret\_error\_responses # handle missing workflows, token overages and auth errors[20][21]  
ap\_atom\_006: use\_client\_libraries # call API via Python or JavaScript SDKs[22][23]  
ap\_atom\_007: monitor\_execution\_status # query status, list executions and download artifacts[24]

Each atomic task encapsulates a single responsibility and can be composed into larger workflows or automated scripts. By separating workflows and enumerating their atoms, teams can implement, test and replace individual steps without impacting adjacent components.

[[1]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-guide.md#L25-L34) [[2]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-guide.md#L39-L46) [[3]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-guide.md#L54-L66) [[4]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-guide.md#L70-L81) [[5]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-guide.md#L84-L96) workflow-guide.md

<https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-guide.md>

[[6]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-enhancements.md#L11-L34) [[7]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-enhancements.md#L37-L67) [[8]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-enhancements.md#L48-L67) [[9]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-enhancements.md#L70-L95) [[10]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-enhancements.md#L97-L133) [[11]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-enhancements.md#L134-L146) [[12]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-enhancements.md#L157-L163) workflow-enhancements.md

<https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/workflow-enhancements.md>

[[13]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/simplified-workflow-guide.md#L1-L41) [[14]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/simplified-workflow-guide.md#L28-L41) simplified-workflow-guide.md

<https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/simplified-workflow-guide.md>

[[15]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L15-L42) [[16]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L25-L42) [[17]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L48-L56) [[18]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L75-L91) [[19]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L98-L114) [[20]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L118-L177) [[21]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L180-L200) [[22]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L207-L245) [[23]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L253-L305) [[24]](https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md#L371-L398) workflow-execution.md

<https://github.com/DICKY1987/CLI_RESTART/blob/96d267ac92f1fc845dd1dc5614b49b387bbd4372/docs/api/examples/workflow-execution.md>