ChatGPT

Engine and Tool Execution Flows in Mosychlos

Mosychlos uses a **chained engine orchestration** approach to mirror FinRobot's multi-layer analysis workflow [1] [2]. Each engine corresponds to a phase (Data, Function, Multi-Agent, Report) and runs sequentially in a single AI session, sharing one context state (the `SharedBag`) [1] [3]. Below, **Figure 1** illustrates the chain from the **Data Gathering Engine** through to the **Report Generation Engine**, with each engine's output feeding into the next. All engines operate on the *same* accumulated context (note: no resetting between engines) [1] [4], ensuring that data collected in earlier stages informs later analyses.

[1] [2]

```
flowchart LR
    Engine1[Data Gathering Engine]:::data -- "Financial data & context" -->
Engine2[Financial Analysis Engine]:::function
    Engine2 -- "Analytical insights" --> Engine3[Investment Committee
Engine]:::multiagent
    Engine3 -- "Final decision" --> Engine4[Report Generation Engine]:::report
    %% All engines share a common SharedBag context throughout
```

Figure 1: Sequential Engine Chain. The **EngineOrchestrator** runs engines in order (Layer 1 Data → Layer 2 Functions → Layer 3 Multi-Agent → Layer 4 Reports) [2]. Each engine adds its results to the shared context for the next engine [3]. This achieves a single unified AI session across all stages, accumulating knowledge and state without resetting [1].

Each engine may internally execute *iterative tool-call cycles* when performing batch analyses. **Figure 2** shows the **Batch Engine Template Method** flow, which handles multi-step tool interactions and looping via hooks [5] [6]. Upon execution, the engine prepares an initial prompt and job (iteration 0), then enters a loop of submitting batch requests to the AI, waiting for results, processing outputs, and deciding whether to continue to another iteration (the **NEXT_PATTERN** transition) [7] [8]. If the AI's response includes tool calls, those are processed via a `ProcessToolResult` hook before generating follow-up jobs [7]. The loop continues until the engine's hooks indicate completion (no further `nextJobs`) [9] [10], after which final results are stored.

[5] [11]

```
flowchart TD
    A[Execute Called] --> B[GetInitialPrompt Hook]
    B --> C[Initialize First BatchJob]
    C --> D{Iteration Loop}
```

```
    D --> E[PreIteration Hook]
    E --> F[Submit Batch to AI]
    F --> G[Wait for Completion]
    G --> H[Process Results]
    H --> I{Tool Calls?}
    I -- Yes --> J[ProcessToolResult Hook]:::tool
    I -- No --> K[ProcessFinalResult Hook]
    J --> L[Generate Next Jobs]
    K --> L
    L --> M[PostIteration Hook]
    M --> N{Continue?}
    N -- Yes --> D
    N -- No --> O[Store Final Results]
    O --> P[Complete]
```

Figure 2: Batch Engine Iterative Flow. A batch-type engine (e.g. for risk analysis) iterates through multiple rounds of AI processing. On each round it may call external tools (if the AI's response requests it), handle those tool results (`ProcessToolResult`), then decide whether to continue [7]. The `ShouldContinueIteration` hook (not shown) breaks the loop when criteria are met, leading to final result storage [9] [10].

Mosychlos can also run **parallel flows** within an engine to simulate multi-expert deliberation. For example, the **InvestmentCommitteeEngine** (Layer 3) orchestrates multiple personas (analyst roles) in parallel to provide diverse perspectives [12] [13]. **Figure 3** depicts a **leader-coordinator pattern**: the committee engine acts as a coordinator that spawns several specialist analyst "agents" simultaneously, then aggregates their insights into a synthesized recommendation [12]. This parallels FinRobot's multi-agent workflow where a chairperson agent coordinates discussion among expert agents [14] [15]. In Mosychlos, however, this is implemented more simply by prompting for multiple perspectives in one engine call (as the AI can handle multiple roles in one context) [12]. The diagram illustrates the logical parallelism: the **Investment Committee Engine** dispatches analysis tasks to, say, a Financial Analyst, Quantitative Analyst, Market Analyst, and Portfolio Manager in parallel, then combines their findings into a final decision.

[16] [14]

```
flowchart LR
    CE[Investment Committee Engine]:::multiagent -->|spawn| FA(Financial Analyst
Persona)
    CE -->|spawn| QA(Quantitative Analyst Persona)
    CE -->|spawn| MA(Market Analyst Persona)
    CE -->|spawn| PM(Portfolio Manager Persona)
    FA -->|insight| SYNT[Synthesis & Recommendation]
    QA -->|insight| SYNT
    MA -->|insight| SYNT
```

```
    PM -->|insight| SYNT
    SYNT -->|final output| OUT[Committee Decision]:::report
```

Figure 3: Parallel Multi-Persona Flow. The **committee engine** coordinates multiple persona sub-tasks in parallel (illustrated as separate analyst roles) and then synthesizes the results. This achieves a multi-expert analysis within one engine's execution [12]. (In practice, the LLM handles this by addressing each perspective in a single prompt, but conceptually it parallels having multiple agents run concurrently.)

Throughout each engine's execution, Mosychlos leverages an integrated **tool execution** mechanism. Engines don't call external APIs directly; instead, the AI client (LLM session) uses *function calling* to invoke tools when needed [6]. **Figure 4** shows how an engine delegates data-fetching or computation to tools: the engine provides the AI with a set of available tools (constraints) and a prompt; the AI may then issue a tool call (or multiple calls in parallel) via the `ToolConsumer` interface [17] [18]. Each tool (e.g., a data API or analytic function) executes and returns results back into the AI's context, after which the AI continues answering with the new information [6]. Mosychlos's base engine infrastructure uniformly handles these tool calls – the engine itself only defines *what* it needs, while the base engine and AiClient manage *how* to invoke the tool and incorporate results [6] [19]. Notably, with GPT-5 the system even supports *parallel tool calls* (concurrent execution of multiple tools) to speed up complex queries [20]. After the AI has gathered required data via tools and completed its reasoning, it returns a final answer, which the engine stores into the SharedBag (for use by subsequent engines or reporting) [21] [22].

[6] [20]

```
flowchart TD
    EngineStart([Engine Execution Start]) --> LLM[AI Client Session]:::function
    LLM -->|Call| Tool1[Tool A]:::tool
    LLM -->|Call| Tool2[Tool B]:::tool
    Tool1 -->|Result| LLM
    Tool2 -->|Result| LLM
    LLM --> Answer[AI Final Answer]
    Answer --> EngineEnd([Engine Stores Result])
```

Figure 4: Tool Calling within an Engine. The engine's AI session can invoke external **Tools** (functions/APIs) as needed. In this example the AI calls **Tool A** and **Tool B** (possibly in parallel) [20], receives their outputs, and then produces the final answer. The engine then stores the result (e.g. in the SharedBag) for downstream use. Mosychlos' base engine automatically handles tool invocation and integration of results [6] – the engine logic only provides the prompt and available tools.

Finally, **Figure 5** integrates these flows into one comprehensive diagram of the **Mosychlos analysis pipeline**. Starting from a user query, the pipeline proceeds through **data gathering**, **analysis**, **investment committee review**, and **report generation**, corresponding to FinRobot's four architecture layers [23] [14].

At each stage, the engine may leverage multiple tools (blue parallelograms) or personas (orange circles) as described above. The diagram is color-coded by phase: **blue** for data layer, **green** for function layer, **orange** for multi-agent layer, **purple** for reporting layer, and **gray** for external tools. This end-to-end view shows how Mosychlos chains together engines and tools to produce an institutional-grade financial report from a single prompt [24] [25].

[23] [26]

```
flowchart TD
    subgraph Mosychlos Full Pipeline
      direction LR
      Start([User Query]) --> DE[Data Engine]:::data
      DE -- consolidates data --> AE[Analysis Engine]:::function
      AE -- insights --> CE[Committee Engine]:::multiagent
      CE -- recommendation --> RE[Report Engine]:::report
      RE --> Output([Professional Report]):::report

      %% Tools used by Data Engine (examples)
      DE -.-> T1[SEC Filings API]:::tool
      DE -.-> T2[Market Data API]:::tool
      DE -.-> T3[News Sentiment API]:::tool

      %% Tools used by Analysis Engine (examples)
      AE -.-> T4[Financial Ratios Tool]:::tool
      AE -.-> T5[Valuation Models]:::tool

      %% Parallel personas in Committee Engine
      CE -.-> P1(Financial Analyst)
      CE -.-> P2(Quant Analyst)
      CE -.-> P3(Market Analyst)
      CE -.-> P4(Portfolio Manager)
    end

    subgraph Legend[Legend]
      direction TB
      LData[Data Layer]:::data
      LFunc[Function Layer]:::function
      LMulti[Multi-Agent Layer]:::multiagent
      LRep[Report Layer]:::report
      LTool[Tool/External API]:::tool
    end
```

Figure 5: Complete Engine & Tool Execution Flow (Mosychlos Pipeline). The end-to-end orchestration from input query to final report. Color coding denotes the phase/layer of each component (see legend). The **Data Engine** (blue) gathers information using various data source tools [27] [28]. The **Analysis Engine** (green) performs in-depth computations, possibly invoking analytic tools. The **Committee Engine** (orange)

incorporates multi-perspective analysis (illustrated by multiple specialist agents) [14]. Finally, the **Report Engine** (purple) compiles everything into a professional report. Mosychlos's architecture thus achieves FinRobot's multi-step analytical process via a chain of engines augmented by tool integrations [26] [24], all within a unified AI-driven workflow.

---

[1] [2] [3] [4] [12] [13] [21] [22] [26] [27] [28] engine-chaining-single-context-analysis.md
https://github.com/amaurybrisou/mosychlos/blob/dde6b2005935822034e19a8bb344e7b7f5278f13/docs/engine-chaining-single-context-analysis.md

[5] [7] [8] batch-engine-template-method-pattern.md
https://github.com/amaurybrisou/mosychlos/blob/dde6b2005935822034e19a8bb344e7b7f5278f13/docs/batch-engine-template-method-pattern.md

[6] [19] README.md
https://github.com/amaurybrisou/mosychlos/blob/dde6b2005935822034e19a8bb344e7b7f5278f13/internal/engine/base/README.md

[9] [10] [11] [17] [18] batch_engine.go
https://github.com/amaurybrisou/mosychlos/blob/dde6b2005935822034e19a8bb344e7b7f5278f13/internal/engine/base/batch_engine.go

[14] [15] [23] [24] [25] finrobot-architecture-reference.md
https://github.com/amaurybrisou/mosychlos/blob/dde6b2005935822034e19a8bb344e7b7f5278f13/docs/finrobot-architecture-reference.md

[16] finrobot-enhancement-plan.md
https://github.com/amaurybrisou/mosychlos/blob/dde6b2005935822034e19a8bb344e7b7f5278f13/docs/finrobot-enhancement-plan.md

[20] gpt-5.md
https://github.com/amaurybrisou/mosychlos/blob/dde6b2005935822034e19a8bb344e7b7f5278f13/docs/gpt-5.md