

AI AGENT PRODUCT FACTORY: MODULE CATALOG

Complete Catalog of all 40 Production Modules

Repository: <https://github.com/sh15aa/agentreusable> • Install: `uv add git+https://github.com/sh15aa/agentreusable.git`

Factory Modules (1)

Master Agent Factory [\[factory.master\]](#) ● STABLE

Path: `agent_factory/factory.py`

Description: Master builder for assembling full production agent instances from identity, gateway, memory, tools, and security layers.

Core Exports: `AgentFactory`, `AgentInstance`

```
from agent_factory.factory import AgentFactory
agent = AgentFactory.create_agent(config, manifest, gateway=gateway, tools=tool_registry)
```

Core Modules (8)

Agent Identity & Configuration [\[core.identity\]](#) ● STABLE

Path: `agent_factory/core/identity.py`

Description: Agent identity specifications, manifests, environment configurations, and cost/retry policies.

Core Exports: `AgentIdentity`, `AgentConfig`, `AgentManifest`, `AgentStatus`, `AgentEnvironment`

```
from agent_factory.core.identity import AgentIdentity, AgentConfig, AgentManifest
identity = AgentIdentity(agent_id='agent-01', name='Researcher')
```

LLM Gateway & Multi-Provider Router [\[core.gateway\]](#) ● STABLE

Path: `agent_factory/core/gateway.py`

Description: Unified LLM gateway with model routing by task complexity, multi-provider fallbacks, streaming, and tool call normalization.

Core Exports: `LLMGateway`, `ModelRouter`, `LLMRequest`, `LLMResponse`, `TaskComplexity`, `MockProvider`, `OpenAICompatibleProvider`

```
from agent_factory.core.gateway import LLMGateway, LLMRequest, TaskComplexity
gateway = LLMGateway()
res = await gateway.complete(LLMRequest(messages=[...], complexity=TaskComplexity.SIMPLE))
```

Token Tracker & Cost Economics [\[core.cost\]](#) ● STABLE

Path: `agent_factory/core/cost.py`

Description: Token usage tracking, multi-model pricing economics, per-run cost recording, and budget limit enforcement.

Core Exports: `CostManager`, `TokenUsage`, `ModelPricing`, `ExecutionCostRecord`

```
from agent_factory.core.cost import CostManager
cost_mgr = CostManager()
cost_mgr.record_usage('gpt-4o', prompt_tokens=150, completion_tokens=50)
```

Prompt Manager & Injection Defense [\[core.prompt\]](#) ● STABLE

Path: agent_factory/core/prompt.py

Description: Dynamic prompt template compilation, versioning, canary tokens for system prompt leakage detection, and prompt injection filtering.

Core Exports: PromptManager, PromptTemplate, PromptInjectionDefense, PromptVersion

```
from agent_factory.core.prompt import PromptManager, PromptInjectionDefense
prompt_mgr = PromptManager()
defense = PromptInjectionDefense()
```

Reliability, Loop Detection & Idempotency [\[core.reliability\]](#) ● STABLE

Path: agent_factory/core/reliability.py

Description: Exponential backoff retries, cycle/loop detection, execution resource bounds, and idempotency guarantees.

Core Exports: RetryPolicy, LoopDetector, ResourceLimits, IdempotencyManager

```
from agent_factory.core.reliability import LoopDetector, ResourceLimits
detector = LoopDetector(threshold=3)
detector.record_step('action_hash')
```

Security, RBAC & Secret Isolation [\[core.security\]](#) ● STABLE

Path: agent_factory/core/security.py

Description: Role-based access control, tool risk categorization (LOW, MEDIUM, HIGH, CRITICAL), SSRF IP/domain validation, and zero-leak secret isolation.

Core Exports: RiskLevel, TenantContext, ToolPermission, SSRFValidator, SecretStore

```
from agent_factory.core.security import RiskLevel, SSRFValidator, SecretStore
SSRFValidator.validate_url('https://api.github.com')
```

Agent Execution State & Persistence [\[core.state\]](#) ● STABLE

Path: agent_factory/core/state.py

Description: Execution context state machine, step tracking, intermediate results, and checkpoint persistence.

Core Exports: ExecutionContext, AgentStateEnum, StepResult, StepStatus

```
from agent_factory.core.state import ExecutionContext
ctx = ExecutionContext(run_id='run-123')
```

Agent Versioning & Canary Rollout [\[core.lifecycle\]](#) ● STABLE

Path: agent_factory/core/lifecycle.py

Description: Semantic agent version management, canary traffic splitting, and automated rollback on regression.

Core Exports: CanaryDeploymentManager, VersionedAgentRecord, DeploymentStatus

```
from agent_factory.core.lifecycle import CanaryDeploymentManager
canary = CanaryDeploymentManager()
```

Memory Modules (6)

Conversation Engine & Multi-turn Threads [\[memory.conversation\]](#) ● STABLE

Path: agent_factory/memory/conversation.py

Description: Multi-turn conversation management, message history formatting, and automated context compaction for large dialogues.

Core Exports: ConversationEngine, ConversationMessage, ConversationThread

```
from agent_factory.memory.conversation import ConversationEngine
conv = ConversationEngine()
conv.add_user_message('thread-1', 'Hello')
```

Working Memory & Scratchpad [\[memory.working\]](#) ● STABLE

Path: agent_factory/memory/working.py

Description: Short-term scratchpad memory storing active goals, discovered findings, completed steps, and task scratchpad notes.

Core Exports: WorkingMemory

```
from agent_factory.memory.working import WorkingMemory
wm = WorkingMemory(objective='Analyze data')
wm.add_finding('Key insight found')
```

Semantic Memory & Vector Retrieval [\[memory.semantic\]](#) ● STABLE

Path: agent_factory/memory/semantic.py

Description: In-memory semantic vector store, text chunking, and cosine similarity ranking for agent knowledge.

Core Exports: SemanticMemory, MemoryChunk, SimpleEmbeddingProvider, cosine_similarity

```
from agent_factory.memory.semantic import SemanticMemory
sm = SemanticMemory()
await sm.store('text document', metadata={'source': 'doc1'})
results = await sm.search('query', top_k=3)
```

Episodic Memory & Experience Learning [\[memory.episodic\]](#) ● STABLE

Path: agent_factory/memory/episodic.py

Description: Stores historical task execution episodes, failures, successes, and derived lessons learned for future task improvement.

Core Exports: EpisodicMemory, Episode

```
from agent_factory.memory.episodic import EpisodicMemory, Episode
em = EpisodicMemory()
em.record_episode(Episode(task='deploy', outcome='success', lessons=['always check env vars']))
```

Hybrid RAG Engine (BM25 + Dense Vectors) [\[memory.rag\]](#) ● STABLE

Path: agent_factory/memory/rag.py

Description: Hybrid retrieval augmented generation combining lexical BM25 indexing with dense semantic vector search and reciprocal rank fusion.

Core Exports: RAGEngine, BM25Index, KnowledgeBase, RAGSearchResult

```
from agent_factory.memory.rag import RAGEngine
rag = RAGEngine()
await rag.ingest_documents([{'content': 'AI agent guide', 'id': 'doc1'}])
results = await rag.query('agent architecture')
```

Memory Governance, PII Redaction & Context Builder [\[memory.governance\]](#) ● STABLE

Path: agent_factory/memory/governance.py

Description: PII scrubbing (emails, credit cards, phones, SSNs), TTL retention pruning, and assembled context token budgeting.

Core Exports: MemoryGovernance, ContextBuilder

```
from agent_factory.memory.governance import MemoryGovernance, ContextBuilder
scrubbed = MemoryGovernance.redact_pii('Contact john@example.com')
```

Tools Modules (7)

Central Tool Registry & Dispatcher [\[tools.registry\]](#) ● STABLE

Path: agent_factory/tools/registry.py

Description: Central registry registering tools, generating standardized JSON tool schemas for LLMs, checking permissions, and dispatching execution.

Core Exports: ToolRegistry

```
from agent_factory.tools.registry import ToolRegistry
registry = ToolRegistry()
registry.register(my_tool)
schemas = registry.get_schemas()
```

Base Tool Contract & Specification [\[tools.base\]](#) ● STABLE

Path: agent_factory/tools/base.py

Description: Abstract base class for all tools with JSON schema extraction, risk levels, and standardized ToolExecutionResult wrapper.

Core Exports: BaseTool, ToolExecutionResult

```
from agent_factory.tools.base import BaseTool, ToolExecutionResult
class MyTool(BaseTool): ...
```

Human Approval & HITL Workflows [\[tools.approval\]](#) ● STABLE

Path: agent_factory/tools/approval.py

Description: Human-in-the-loop approval queue intercepting HIGH and CRITICAL risk tool executions before dispatch.

Core Exports: HumanApprovalQueue, ApprovalRequest, ApprovalStatus

```
from agent_factory.tools.approval import HumanApprovalQueue
queue = HumanApprovalQueue()
req = queue.request_approval(tool_name='delete_database', args={...})
```

Web Search Tool [\[tools.builtins.search\]](#) ● STABLE

Path: agent_factory/tools/builtins/search.py

Description: Structured search tool for external information gathering with query parameters, filtering, and result summaries.

Core Exports: WebSearchTool, SearchInput, SearchResultItem, SearchOutput

```
from agent_factory.tools.builtins.search import WebSearchTool
search = WebSearchTool()
result = await search.execute(query='latest python features')
```

Browser Action Tool with SSRF Defense [\[tools.builtins.browser\]](#) ● STABLE

Path: agent_factory/tools/builtins/browser.py

Description: Web browser automation tool with URL sanitization, domain allow/deny list, and private network SSRF defenses.

Core Exports: BrowserTool, BrowserActionInput, BrowserActionOutput

```
from agent_factory.tools.builtins.browser import BrowserTool
browser = BrowserTool()
page = await browser.execute(url='https://example.com')
```

Hardened Python Sandbox Tool [\[tools.builtins.code_exec\]](#) ● STABLE

Path: agent_factory/tools/builtins/code_exec.py

Description: AST-inspected, isolated Python code execution tool restricting dangerous builtins, modules, and excessive CPU/memory runtime.

Core Exports: PythonSandboxTool, CodeExecutionInput, CodeExecutionOutput

```
from agent_factory.tools.builtins.code_exec import PythonSandboxTool
sandbox = PythonSandboxTool()
res = await sandbox.execute(code='result = [x**2 for x in range(10)]')
```

Document & File Processing Tool [\[tools.builtins.files\]](#) ● STABLE

Path: agent_factory/tools/builtins/files.py

Description: File parser and reader supporting text, markdown, CSV, JSON, and PDF text extraction with strict path traversal prevention.

Core Exports: FileProcessingTool, FileParseInput, FileParseOutput

```
from agent_factory.tools.builtins.files import FileProcessingTool
files = FileProcessingTool()
content = await files.execute(file_path='data/report.json')
```

Orchestration Modules (4)

Planning Engine & Task Decomposition [\[orchestration.planner\]](#) ● STABLE

Path: agent_factory/orchestration/planner.py

Description: Breaks complex user goals into sequential/parallel dependency DAG steps, handles dynamic re-planning on step failures.

Core Exports: PlanningEngine, Plan, PlanStep

```
from agent_factory.orchestration.planner import PlanningEngine
planner = PlanningEngine(gateway=gateway)
plan = await planner.create_plan('Build customer support workflow')
```

Workflow Execution Engine (DAG Orchestrator) [\[orchestration.workflow\]](#) ● STABLE

Path: agent_factory/orchestration/workflow.py

Description: Directed acyclic graph execution engine supporting parallel branches, dependency resolution, error traps, and conditional routing.

Core Exports: WorkflowEngine

```
from agent_factory.orchestration.workflow import WorkflowEngine
engine = WorkflowEngine()
res = await engine.execute_dag(dag_definition)
```

Multi-Agent Orchestration & Supervisor [\[orchestration.multi_agent\]](#) ● STABLE

Path: agent_factory/orchestration/multi_agent.py

Description: Agent-to-agent communication protocol, message passing, peer collaboration, and hierarchical supervisor orchestration.

Core Exports: MultiAgentOrchestrator, SupervisorAgent, AgentMessage, AgentMessageType

```
from agent_factory.orchestration.multi_agent import MultiAgentOrchestrator, SupervisorAgent
supervisor = SupervisorAgent(workers=[worker1, worker2])
```

Event Bus, Trigger Engine & Job Queue [\[orchestration.events\]](#) ● STABLE

Path: agent_factory/orchestration/events.py

Description: Asynchronous publish-subscribe event bus, event triggers for automated agent dispatch, and background job queuing.

Core Exports: EventBus, BackgroundJobQueue, AgentEvent

```
from agent_factory.orchestration.events import EventBus, AgentEvent
bus = EventBus()
bus.subscribe('order.created', handler_fn)
```

Validation Modules (3)

Structured Output Engine & Schema Enforcer [\[validation.structured\]](#) ● STABLE

Path: agent_factory/validation/structured.py

Description: Extracts and validates JSON outputs against Pydantic models, automatically repairing broken JSON and syntax mismatches.

Core Exports: StructuredOutputEngine

```
from agent_factory.validation.structured import StructuredOutputEngine
validator = StructuredOutputEngine()
model_obj = validator.parse_and_validate(raw_llm_text, target_pydantic_cls)
```

Self-Verification & Fact Checking [\[validation.verification\]](#) ● STABLE

Path: agent_factory/validation/verification.py

Description: Multi-pass verification validating LLM responses against source material, checking hallucinations and factual alignment.

Core Exports: SelfVerificationEngine, VerificationResult, FactVerificationItem

```
from agent_factory.validation.verification import SelfVerificationEngine
verifier = SelfVerificationEngine(gateway=gateway)
result = await verifier.verify_claims(answer, context_docs)
```

Citation Engine & Source Provenance [\[validation.citations\]](#) ● STABLE

Path: agent_factory/validation/citations.py

Description: Maps claims and generated statements back to verified document source IDs, line ranges, or URLs.

Core Exports: CitationEngine, Citation

```
from agent_factory.validation.citations import CitationEngine
citations = CitationEngine()
annotated = citations.link_citations(generated_text, retrieved_sources)
```

Observability Modules (3)

Distributed Tracing & Span Trees [\[observability.tracer\]](#) ● STABLE

Path: agent_factory/observability/tracer.py

Description: Hierarchical span tracing for agent thought loops, LLM calls, tool executions, and timing diagnostics.

Core Exports: Tracer, Span

```
from agent_factory.observability.tracer import Tracer
tracer = Tracer()
with tracer.start_span('agent_step'): ...
```

Security Audit Logging [\[observability.audit\]](#) ● STABLE

Path: agent_factory/observability/audit.py

Description: Tamper-evident audit logging for tool executions, privilege elevations, policy rejections, and access logs.

Core Exports: AuditLogger, AuditRecord

```
from agent_factory.observability.audit import AuditLogger
audit = AuditLogger()
audit.log_event(action='tool_call', user_id='usr_123', risk='MEDIUM')
```

Operational Metrics & Analytics [\[observability.metrics\]](#) ● STABLE

Path: agent_factory/observability/metrics.py

Description: Tracks latency distributions, token usage, tool success/failure ratios, and cost velocity.

Core Exports: AgentAnalytics, AgentAnalyticsSnapshot

```
from agent_factory.observability.metrics import AgentAnalytics
analytics = AgentAnalytics()
snapshot = analytics.get_snapshot()
```

Evaluation Modules (3)

Golden Evaluation Datasets & Test Cases [\[evaluation.dataset\]](#) ● STABLE

Path: agent_factory/evaluation/dataset.py

Description: Golden test case schema, expected outputs, ground truth rubrics, and dataset serialization.

Core Exports: GoldenDataset, TestCase

```
from agent_factory.evaluation.dataset import GoldenDataset, TestCase
ds = GoldenDataset(name='qa_tests')
ds.add_test(TestCase(prompt='...', expected='...'))
```

LLM-as-Judge & Rubric Scoring [\[evaluation.judge\]](#) ● STABLE

Path: agent_factory/evaluation/judge.py

Description: Automated qualitative evaluation scoring agent outputs on accuracy, clarity, safety, and helpfulness (1-5 scale).

Core Exports: LLMAJudge, JudgeScorecard

```
from agent_factory.evaluation.judge import LLMAJudge
judge = LLMAJudge(gateway=gateway)
scorecard = await judge.evaluate(prompt, response, ground_truth)
```

Regression Benchmark Runner [\[evaluation.regression\]](#) ● STABLE

Path: agent_factory/evaluation/regression.py

Description: Automated benchmark suite running GoldenDatasets against agent instances and generating pass/fail reports.

Core Exports: RegressionRunner, BenchmarkReport, CaseEvaluationReport

```
from agent_factory.evaluation.regression import RegressionRunner
runner = RegressionRunner(agent_instance)
report = await runner.run_benchmark(golden_dataset)
```

Api Modules (2)

Production FastAPI Agent Server [\[api.server\]](#) ● STABLE

Path: agent_factory/api/server.py

Description: REST API, Server-Sent Events (SSE) streaming endpoint, and webhook callback integration for agents.

Core Exports: RunAgentRequest, RunAgentResponse, ResolveApprovalRequest

```
import uvicorn
# Run with: uvicorn agent_factory.api.server:app --port 8000
```

API Auth, Tenant Resolution & HMAC Webhooks [\[api.auth\]](#) ● STABLE

Path: agent_factory/api/auth.py

Description: Bearer token authentication, multi-tenant isolation, rate limiting, and HMAC signature verification for webhooks.

Core Exports: RateLimiter, verify_webhook_signature

```
from agent_factory.api.auth import RateLimiter, verify_webhook_signature
limiter = RateLimiter(requests_per_minute=60)
```

Templates Modules (3)

Deep Research Agent Template [\[templates.research_agent\]](#) ● STABLE

Path: templates/research_agent.py

Description: Pre-configured ready-to-run Deep Web Search, Fact-Checking, and Citation Research Agent.

Core Exports: create_research_agent

```
from templates.research_agent import create_research_agent
agent = create_research_agent()
result = await agent.run('Investigate latest quantum computing breakthroughs')
```

Content Strategy Agent Template [\[templates.content_agent\]](#) ● STABLE

Path: templates/content_agent.py

Description: Pre-configured Content Strategy, Tone-of-Voice, and SEO Writing Agent.

Core Exports: create_content_agent

```
from templates.content_agent import create_content_agent
agent = create_content_agent()
result = await agent.run('Draft product announcement blog post')
```

Data Analysis & Coding Agent Template [\[templates.coding_agent\]](#) ● STABLE

Path: templates/coding_agent.py

Description: Pre-configured Python Sandbox, Data Analysis, and Code Execution Agent.

Core Exports: create_coding_agent

```
from templates.coding_agent import create_coding_agent
agent = create_coding_agent()
result = await agent.run('Calculate summary statistics from dataset')
```