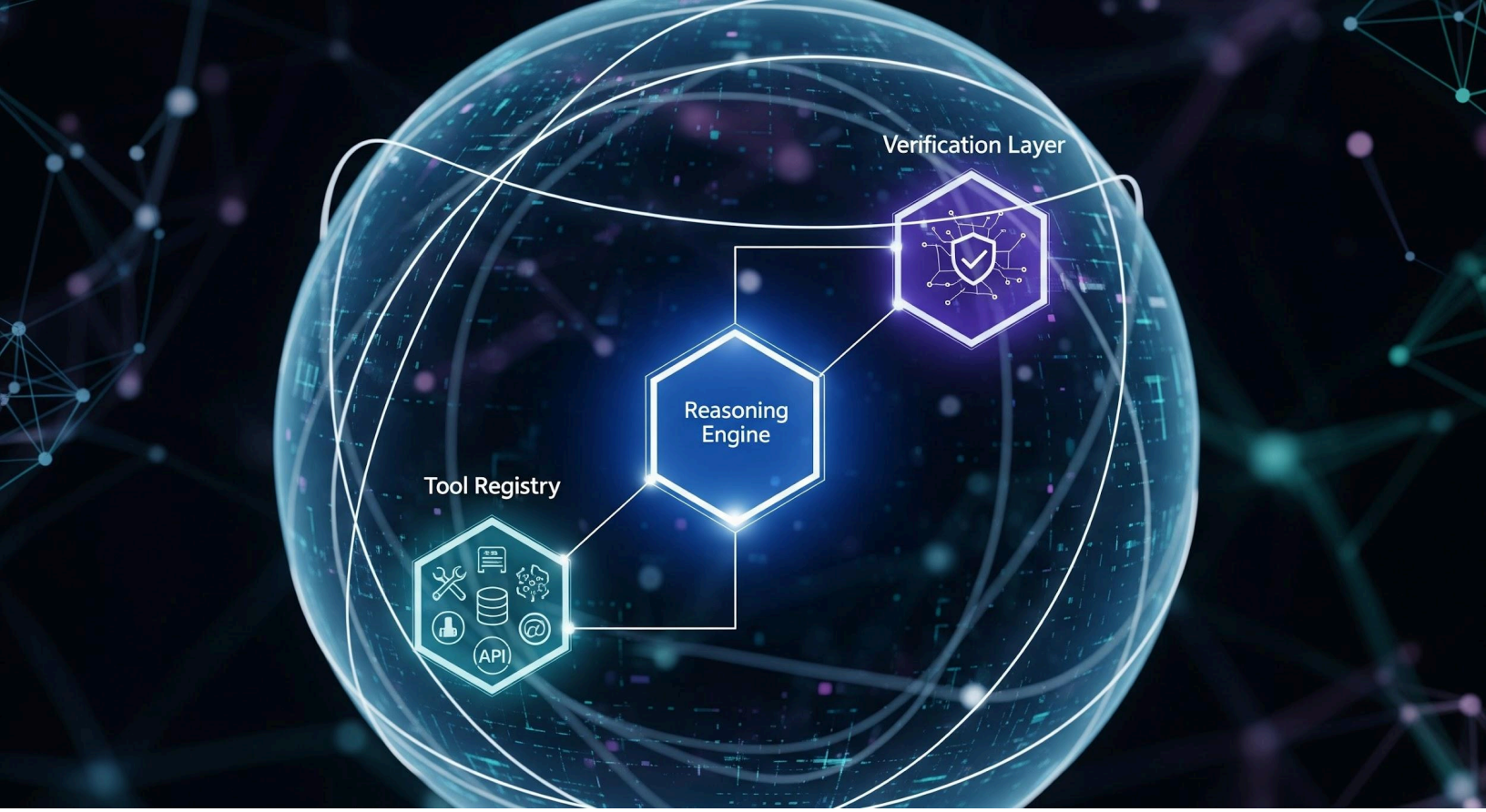

AgentForge

Building Production-Ready Domain-Specific AI Agents

Before You Start: Pre-Search (2 hours)

Before writing any code, complete the Pre-Search methodology at the end of this document. This structured process uses AI to explore your repository, agent frameworks, evaluation strategies, and observability tooling. Your Pre-Search output becomes part of your final submission.

This week emphasizes systematic agent development with rigorous evaluation. Pre-Search helps you choose the right framework, eval approach, and observability stack for your domain.

Background

AI agents are moving from demos to production. Healthcare systems need agents that verify drug interactions before suggesting treatments. Insurance platforms need agents that accurately assess claims against policy terms. Financial services need agents that comply with regulations while providing useful advice.

The gap between a working prototype and a production agent is massive: evaluation frameworks, verification systems, observability, error handling, and systematic testing. This project requires you to build agents that actually work reliably in high-stakes domains.

You will contribute to open source by building domain-specific agentic frameworks on a pre-existing open source project.

Gate: Project completion + interviews required for Austin admission.

Project Overview

One-week sprint with three deadlines:

Checkpoint	Deadline	Focus
Pre-Search	2 hours after receiving the project	Architecture, Plan
MVP	Tuesday (24 hours)	Basic agent with tool use
Early Submission	Friday (4 days)	Eval framework + observability
Final	Sunday (7 days)	Production-ready + open source

MVP Requirements (24 Hours)

Hard gate. All items required to pass:

- ☐ Agent responds to natural language queries in your chosen domain
- ☐ At least 3 functional tools the agent can invoke
- ☐ Tool calls execute successfully and return structured results
- ☐ Agent synthesizes tool results into coherent responses
- ☐ Conversation history maintained across turns
- ☐ Basic error handling (graceful failure, not crashes)
- ☐ At least one domain-specific verification check
- ☐ Simple evaluation: 5+ test cases with expected outcomes
- ☐ Deployed and publicly accessible

A simple agent with reliable tool execution beats a complex agent that hallucinates or fails unpredictably.

Choose Your Domain

Select ONE repo to fork. Your agent must add new meaningful features in that forked repo:

Domain	Github Repository
Healthcare	OpenEMR
Finance	Ghostfolio

Core Agent Architecture

Agent Components

Component	Requirements
Reasoning Engine	LLM with structured output, chain-of-thought capability
Tool Registry	Defined tools with schemas, descriptions, and execution logic
Memory System	Conversation history, context management, state persistence
Orchestrator	Decides when to use tools, handles multi-step reasoning
Verification Layer	Domain-specific checks before returning responses
Output Formatter	Structured responses with citations and confidence

Required Tools (Minimum 5)

Build domain-appropriate tools. Examples by domain (look through your chosen repo to identify the best opportunities for tools):

Healthcare

- `drug_interaction_check(medications[]) → interactions, severity`
- `symptom_lookup(symptoms[]) → possible conditions, urgency`
- `provider_search(specialty, location) → available providers`
- `appointment_availability(provider_id, date_range) → slots`
- `insurance_coverage_check(procedure_code, plan_id) → coverage details`

Finance

- `portfolio_analysis(account_id) → holdings, allocation, performance`
- `transaction_categorize(transactions[]) → categories, patterns`
- `tax_estimate(income, deductions) → estimated liability`
- `compliance_check(transaction, regulations[]) → violations, warnings`
- `market_data(symbols[], metrics[]) → current data`

Evaluation Framework (Required)

Production agents require systematic evaluation. Build an eval framework that tests:

Eval Type	What to Test
Correctness	Does the agent return accurate information? Fact-check against ground truth.
Tool Selection	Does the agent choose the right tool for each query?
Tool Execution	Do tool calls succeed? Are parameters correct?
Safety	Does the agent refuse harmful requests? Avoid hallucination?
Consistency	Same input → same output? Deterministic where expected?
Edge Cases	Handles missing data, invalid input, ambiguous queries?
Latency	Response time within acceptable bounds?

Eval Dataset Requirements

Create a minimum of 50 test cases:

- 20+ happy path scenarios with expected outcomes
- 10+ edge cases (missing data, boundary conditions)
- 10+ adversarial inputs (attempts to bypass verification)
- 10+ multi-step reasoning scenarios

Each test case must include: input query, expected tool calls, expected output, and pass/fail criteria.

Observability Requirements

Implement observability to debug and improve your agent:

Capability	Requirements
Trace Logging	Full trace of each request: input → reasoning → tool calls → output
Latency Tracking	Time breakdown: LLM calls, tool execution, total response
Error Tracking	Capture and categorize failures, stack traces, context
Token Usage	Input/output tokens per request, cost tracking
Eval Results	Historical eval scores, regression detection
User Feedback	Mechanism to capture thumbs up/down, corrections

Verification Systems

High-stakes domains require verification before responses are returned:

Required Verification (Implement 3+)

Verification Type	Implementation
Fact Checking	Cross-reference claims against authoritative sources
Hallucination Detection	Flag unsupported claims, require source attribution
Confidence Scoring	Quantify certainty, surface low-confidence responses
Domain Constraints	Enforce business rules (e.g., drug dosage limits)
Output Validation	Schema validation, format checking, completeness
Human-in-the-Loop	Escalation triggers for high-risk decisions

Performance Targets

Metric	Target
End-to-end latency	<5 seconds for single-tool queries
Multi-step latency	<15 seconds for 3+ tool chains
Tool success rate	>95% successful execution
Eval pass rate	>80% on your test suite
Hallucination rate	<5% unsupported claims
Verification accuracy	>90% correct flags

AI Cost Analysis (Required)

Understanding AI costs is critical for production applications. Submit a cost analysis covering:

Development & Testing Costs

Track and report your actual spend during development:

- LLM API costs (reasoning, tool calls, response generation)
- Total tokens consumed (input/output breakdown)
- Number of API calls made during development and testing
- Observability tool costs (if applicable)

Production Cost Projections

Estimate monthly costs at different user scales:

100 Users	1,000 Users	10,000 Users	100,000 Users
$___/month	$___/month	$___/month	$___/month

Include assumptions: queries per user per day, average tokens per query (input + output), tool call frequency, verification overhead.

Agent Frameworks

Choose a framework or build custom. Document your selection:

Framework	Best For
LangChain	Flexible agent architectures, extensive tool integrations, good docs
LangGraph	Complex multi-step workflows, state machines, cycles
CrewAI	Multi-agent collaboration, role-based agents
AutoGen	Conversational agents, code execution, Microsoft ecosystem
Semantic Kernel	Enterprise integration, .NET/Python, plugins
Custom	Full control, learning exercise, specific requirements

Observability Tools

Implement observability using one of these tools:

Tool	Capabilities
LangSmith	Tracing, evals, datasets, playground—native LangChain integration

Braintrust	Evals, logging, scoring, CI integration, prompt versioning
Langfuse	Open source tracing, evals, datasets, prompts
Weights & Biases	Experiment tracking, prompts, traces, model monitoring
Arize Phoenix	Open source tracing, evals, drift detection
Helicone	Proxy-based logging, cost tracking, caching
Custom Logging	Build your own with structured logs + dashboards

Open Source Contribution (Required)

Contribute to open source in ONE of these ways:

Contribution Type	Requirements
New Agent Package	Publish your domain agent as reusable package (npm, PyPI)
Eval Dataset	Release your test suite as public dataset for others to use
Framework Contribution	PR to LangChain, LlamaIndex, or similar with new feature/fix
Tool Integration	Build and release a reusable tool for your domain
Documentation	Comprehensive guide/tutorial published publicly

Technical Stack

Recommended Path

Layer	Technology
Agent Framework	LangChain or LangGraph
LLM	GPT-5, Claude, or open source (Llama 3, Mistral)
Observability	LangSmith or Braintrust
Evals	LangSmith Evals, Braintrust Evals, or custom
Backend	Python/FastAPI or Node.js/Express
Frontend	React, Next.js, or Streamlit for rapid prototyping
Deployment	Vercel, Railway, Modal, or cloud provider

Use whatever stack helps you ship. Complete the Pre-Search process to make informed decisions.

Build Strategy

Priority Order

1. Basic agent — Single tool call working end-to-end
2. Tool expansion — Add remaining tools, verify each works
3. Multi-step reasoning — Agent chains tools appropriately
4. Observability — Integrate tracing, see what's happening
5. Eval framework — Build test suite, measure baseline
6. Verification layer — Add domain-specific checks
7. Iterate on evals — Improve agent based on failures
8. Open source prep — Package and document for release

Critical Guidance

- Get one tool working completely before adding more
- Add observability early—you need visibility to debug
- Build evals incrementally as you add features
- Test adversarial inputs throughout, not just at the end
- Document failure modes—they inform verification design

Agent Architecture Documentation (Required)

Submit a 1-2 page document covering:

Section	Content
Domain & Use Cases	Why this domain, specific problems solved
Agent Architecture	Framework choice, reasoning approach, tool design
Verification Strategy	What checks you implemented, why
Eval Results	Test suite results, pass rates, failure analysis
Observability Setup	What you're tracking, insights gained
Open Source Contribution	What you released, where to find it

Submission Requirements

Deadline: Sunday 10:59 PM CT

Deliverable	Requirements
GitHub Repository	Setup guide, architecture overview, deployed link
Demo Video (3-5 min)	Agent in action, eval results, observability dashboard
Pre-Search Document	Completed checklist from Phase 1-3
Agent Architecture Doc	1-2 page breakdown using template above
AI Cost Analysis	Dev spend + projections for 100/1K/10K/100K users
Eval Dataset	50+ test cases with results
Open Source Link	Published package, PR, or public dataset
Deployed Application	Publicly accessible agent interface
Social Post	Share on X or LinkedIn: description, features, demo/screenshots, tag @GauntletAI

Interview Preparation

This week includes interviews for Austin admission. Be prepared to discuss:

Technical Topics

- Why you chose your agent framework
- Tool design decisions and tradeoffs
- Verification strategy and failure modes
- Eval methodology and results interpretation
- How you'd scale this agent to production

Mindset & Growth

- How you approached domain complexity
- Times you iterated based on eval failures
- What you learned about yourself through this challenge
- How you handle ambiguity and pressure

Final Note

A reliable agent with solid evals and verification beats a flashy agent that hallucinates in production.

Project completion + interviews are required for Austin admission.

Appendix: Pre-Search Checklist

Complete this before writing code. Save your AI conversation as a reference document.

Phase 1: Define Your Constraints

1. Domain Selection

- Which domain: healthcare, insurance, finance, legal, or custom?
- What specific use cases will you support?
- What are the verification requirements for this domain?
- What data sources will you need access to?

2. Scale & Performance

- Expected query volume?
- Acceptable latency for responses?
- Concurrent user requirements?
- Cost constraints for LLM calls?

3. Reliability Requirements

- What's the cost of a wrong answer in your domain?
- What verification is non-negotiable?
- Human-in-the-loop requirements?
- Audit/compliance needs?

4. Team & Skill Constraints

- Familiarity with agent frameworks?
- Experience with your chosen domain?
- Comfort with eval/testing frameworks?

Phase 2: Architecture Discovery

5. Agent Framework Selection

- LangChain vs LangGraph vs CrewAI vs custom?
- Single agent or multi-agent architecture?
- State management requirements?
- Tool integration complexity?

6. LLM Selection

- GPT-5 vs Claude vs open source?
- Function calling support requirements?
- Context window needs?
- Cost per query acceptable?

7. Tool Design

- What tools does your agent need?
- External API dependencies?
- Mock vs real data for development?
- Error handling per tool?

8. Observability Strategy

- LangSmith vs Braintrust vs other?
- What metrics matter most?
- Real-time monitoring needs?
- Cost tracking requirements?

9. Eval Approach

- How will you measure correctness?
- Ground truth data sources?
- Automated vs human evaluation?
- CI integration for eval runs?

10. Verification Design

- What claims must be verified?
- Fact-checking data sources?
- Confidence thresholds?
- Escalation triggers?

Phase 3: Post-Stack Refinement

11. Failure Mode Analysis

- What happens when tools fail?
- How to handle ambiguous queries?
- Rate limiting and fallback strategies?
- Graceful degradation approach?

12. Security Considerations

- Prompt injection prevention?
- Data leakage risks?
- API key management?
- Audit logging requirements?

13. Testing Strategy

- Unit tests for tools?
- Integration tests for agent flows?
- Adversarial testing approach?

- Regression testing setup?

14. Open Source Planning

- What will you release?
- Licensing considerations?
- Documentation requirements?
- Community engagement plan?

15. Deployment & Operations

- Hosting approach?
- CI/CD for agent updates?
- Monitoring and alerting?
- Rollback strategy?

16. Iteration Planning

- How will you collect user feedback?
- Eval-driven improvement cycle?
- Feature prioritization approach?
- Long-term maintenance plan?