SEMEAI / PRODUCT ROADMAP

From Evidence Runtime to Agent Skill Marketplace

Detailed execution roadmap v1.0

- ACCOUNT + WORKSPACE
- GENESIS EVOLUTION TRACE
- BENCHMARK AUDIT + ROADMAP
- CODEX IMPROVEMENT LOOP + SKILL FORGE
- OPTIONAL AGENT SKILL MARKETPLACE

Core principle

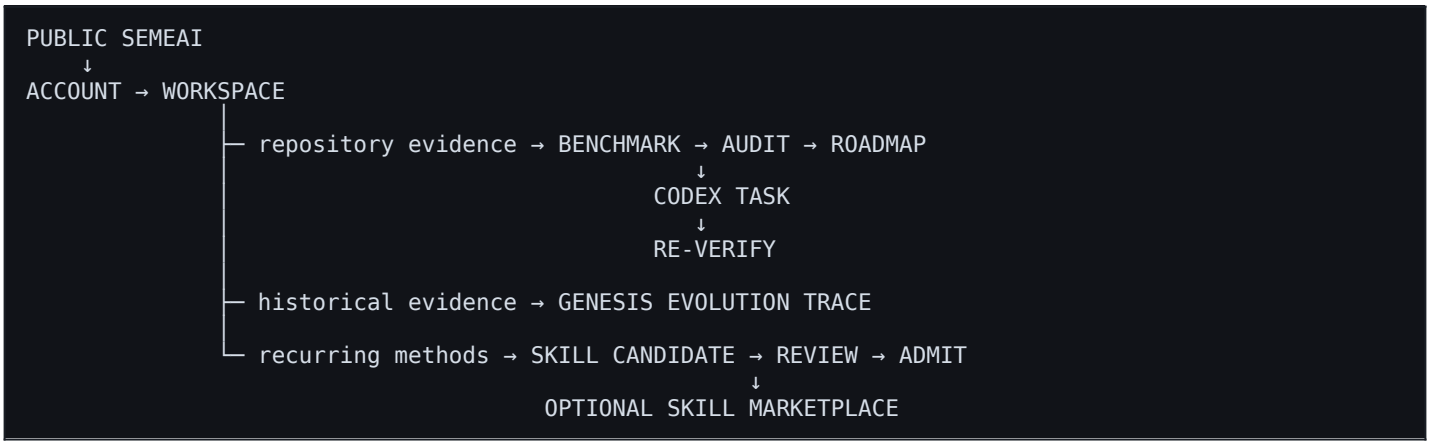
Generation is not release authority.

Status: architecture roadmap / execution authority document
Prepared for staged implementation, validation, and market discovery.

Executive Summary

> **Roadmap thesis**
>
> SemeAI should evolve from a public release-control and evidence system into a product loop where a user can enter a governed workspace, inspect real repositories, receive an evidence-based improvement roadmap, apply bounded changes through Codex, and later extract reusable agent skills. A marketplace for those skills is a possible commercialization layer - not a prerequisite for the core product.

The roadmap is deliberately dependency-driven. Each phase must create evidence required by the next phase. The goal is not to add more surfaces; it is to make the surfaces already built participate in one product loop.



What changes in the product

- The public site stops being the end of the journey. Account and Workspace become the private product spine.
- Genesis becomes an evidence-backed history of the ecosystem rather than only a cinematic myth of origin.
- Benchmark evolves from score + rank into repository audit + evidence roadmap.
- Benchmark results become workspace evidence artifacts tied to repository, commit, capture time, policy, decision, and receipt.
- Codex becomes an execution layer for bounded improvement tasks created from evidence gaps, not a generic “improve my repo” button.
- Skill Forge turns repeated, evidenced workflows into skill candidates. Generation proposes; review/admission authorizes.
- A marketplace can later sell or distribute admitted agent skills, with provenance, compatibility, evidence, versioning, and receipts as the trust layer.

First marketplace case

> **Case 0 - Creator skills generated by Codex**

> Before opening a marketplace to third parties, SemeAI should use its own development history as the first end-to-end case: Codex analyzes first-party repositories, documentation, recurring changes, tests, and release patterns; proposes skill candidates; the creator reviews and admits them; those skills are then used on real work. Only after this loop is repeatable should external listings and paid distribution be considered.

Success definition

Layer	A user can…	Proof required
Workspace	Sign in and enter a real governed context	Backend-backed account/session/workspace identity
Benchmark	Understand why a repository scored as it did	Deterministic evidence categories and receipt
Roadmap	See missing observable evidence and bounded next steps	Gap-to-action mapping without fabricated guarantees
Codex loop	Apply one bounded improvement and re-measure	Diff, tests, new snapshot, comparable score
Skill Forge	Generate and review a reusable method candidate	Evidence references + review/admission record
Marketplace	Evaluate, acquire, and use a skill with provenance	Version, compatibility, evidence, receipt, seller identity

Current State and Product Boundary

This roadmap starts from a system that already has substantial working parts. The execution plan assumes the existing release-control semantics, repository benchmark authority, Engineering Book, public research framing, and visual/motion language remain protected while the missing product spine is added.

Already established

- Public product surfaces: Home, Genesis, Gate, Repository Evidence Benchmark, Engineering Book, Research, Dashboard.
- Release-control principle: generation creates a candidate; release is a separate decision authority.
- User-facing states: SHOW / REVIEW / BLOCK; internal semantics remain bounded by existing SaC/PoR authority.
- Repository benchmark: live public GitHub snapshot, deterministic scoring policy, presentation Gate, rank, receipt.
- Visual language and motion language: graphite/near-black space, restrained gold/cyan/violet semantics, motion tied to system meaning.
- Existing SaaS path: registration, verification, login/session, account, usage, receipts, billing/team capabilities where already supported by backend.

Missing pieces

Missing layer	Why it matters	Roadmap response
Private product spine	Public surfaces have no durable user context	Account + Workspace Foundation
Evidence history	Genesis is aesthetic but not yet a factual ecosystem trace	Historical archive + manifest + Evolution Trace
Actionable benchmark	Score is useful, but users want “what should I improve?”	Audit + evidence roadmap
Workspace evidence persistence	Benchmark result does not yet become user-owned evidence	Evidence artifact backend and Save to Workspace
Closed improvement loop	No bounded measure → change → verify product flow	Codex improvement loop
Reusable method layer	Repeated workflows are not captured as governed artifacts	Skill Forge + registry
Commercial distribution	No trustable channel to sell reusable agent skills	Optional marketplace after internal validation

Non-goals / claims boundary

- No AGI, consciousness, universal safety, or universal code-quality claims.
- No claim that Benchmark certifies security, compliance, correctness, or production readiness.
- No autonomous skill admission: an LLM may generate a candidate, but cannot authorize its own admission.

- No fake multi-workspace behavior, fake activity, fake memories, fake files, or fake metrics.
- No marketplace claim before acquisition, compatibility, versioning, seller identity, and evidence flows actually work.
- No post-Gate mutation that changes an already-authorized decision artifact.

> **Frozen product invariant**
>
> Every new feature must preserve the separation between candidate generation, evidence, authority, decision, and retained audit/receipt. Marketplace commerce, workspace access, payment, popularity, or seller status must never become release authority.

Target Product Architecture

```
SEMEAI.TECH
│
├── PUBLIC
│   ├── Home
│   ├── Genesis / Evolution Trace
│   ├── Gate
│   ├── Benchmark / Repository Audit
│   ├── Engineering Book
│   └── Research
│
├── IDENTITY
│   └── Account
│
├── PRODUCT CONTEXT
│   └── Workspace
│       ├── Conversations
│       ├── Sources
│       ├── Memory
│       ├── Evidence
│       ├── Decisions
│       ├── Receipts
│       └── Skills
│
├── EXECUTION
│   ├── Gate / operator dashboard
│   ├── Codex improvement tasks
│   └── verification / replay
│
└── DISTRIBUTION (later)
    ├── Skill Registry
    └── Agent Skill Marketplace
```

Canonical product loop

```
REPOSITORY
   ↓
CAPTURED SNAPSHOT
   ↓
NORMALIZED EVIDENCE
   ↓
BENCHMARK SCORE + GATE + RECEIPT
   ↓
AUDIT / EVIDENCE GAPS
   ↓
BOUNDED ROADMAP TASK
   ↓
CODEX CHANGE CANDIDATE
   ↓
TEST / VERIFY
   ↓
NEW SNAPSHOT
   ↓
COMPARE DELTA
   ↓
REUSABLE METHOD CANDIDATE
   ↓
SKILL REVIEW / ADMISSION
```

Data ownership boundary

Entity	Owner/context	Persistence expectation	Authority
Account	User identity	Backend	Access only
Workspace	User/project context	Backend	Context boundary only
Repository snapshot	Workspace evidence	Backend + immutable capture metadata	Evidence
Benchmark receipt	Snapshot/policy result	Immutable/retained	Proof of decision trace
Roadmap item	Derived from evidence gap	Backend	Planning artifact
Codex task result	Workspace/change attempt	Git + workspace metadata	Candidate change
Skill candidate	Workspace/repository lineage	Backend + Git artifact	Candidate method
Admitted skill	Registry	Versioned immutable release artifact	Authorized reusable method
Marketplace listing	Seller + admitted skill version	Marketplace DB	Commercial presentation only

04 / EXECUTION ORDER

Roadmap Overview

Phase	Name	Dependency	Exit gate
0	Authority Freeze & Inventory	Current production state	Known clean baseline, contracts documented
1	Account + Workspace Foundation v0.1	Existing auth/account backend	User can enter real workspace
2	Historical Archive + Genesis Manifest	First-party archive + GitHub inventory	Evidence-backed timeline data exists
3	Genesis v03 - Evolution Trace	Phase 2	Data-driven historical surface live
4	Benchmark Audit + Evidence Roadmap	Current Benchmark authority	Users see gaps and bounded next steps
5	Repository Evidence in Workspace	Workspace + Benchmark	Snapshots/receipts persist per workspace
6	Codex Improvement Loop v0.1	Phase 4 + 5	Measure → change → verify works end-to-end
7	Skill Forge v0.1	Phase 6	First creator skill generated, reviewed, admitted, reused
8	Skill Registry + Compatibility	Phase 7	Skills versioned, installable, testable
9	Optional Agent Skill Marketplace	Phase 8 + real demand	External listing/acquisition loop proven
10	Cross-System Integration & Final Polish	Stable product spine	One coherent product, not isolated surfaces
11	Pilot / GTM / Monetization	Working loops	External usage and payment evidence

Recommended rule for sequencing

> **Do not parallelize dependency work**
>
> Visual polish can run in parallel with isolated maintenance, but Workspace, Evidence persistence, Codex loop, Skill Forge, and Marketplace should be sequential. Each later layer needs proof from the previous layer. Skipping the gates will create demo-only surfaces and claim drift.

Indicative 12-week shape (not a commitment)

Weeks	Primary focus	Secondary
1-2	Phase 1: Account + Workspace	Authority audit, no redesign

Weeks	Primary focus	Secondary
3	Phase 2: Archive + manifest	Repository/fork classification
4-5	Phase 3: Genesis v03	Mobile/reduced-motion validation
6	Phase 4: Benchmark Audit + Roadmap	Test with real external repos
7	Phase 5: Save to Workspace	Evidence backend endpoints
8-9	Phase 6: Codex improvement loop	One bounded improvement case
10	Phase 7: Skill Forge	Generate first creator skill
11	Phase 8: Skill Registry	Version/install/compatibility
12	Integration pass	Decide whether marketplace hypothesis is ready

Schedule depends on actual backend complexity and validation failures. Gate progression by evidence, not calendar.

05 / BASELINE

Phase 0 - Authority Freeze & Inventory

Purpose: prevent a new product spine from silently changing benchmark authority, receipt semantics, routing contracts, or deployment behavior.

Deliverables

- Record current local HEAD, origin/master, deployed Pages state, worktree cleanliness, and known production URLs.
- Inventory client/backend methods actually available: register, verify, login/logout, account, usage, receipts, billing, team, keys, provider availability.
- Freeze Benchmark scoring policy, evidence derivation, indicators, Gate semantics, receipt canonicalization/hash, rank semantics, seed/phase logic.
- Document public route IA and identify compatibility redirects that must remain.
- Create a short architecture decision record: Account is identity/access; Workspace is governed context; Dashboard is operator console.

Acceptance gate

Check	Pass condition
Repository baseline	Clean, reproducible HEAD; no unknown local changes
Backend contract	Known endpoints/payloads captured from repo authority
Frozen invariants	Golden benchmark and receipt hashes unchanged
Scope	No Home/Genesis/Gate/Book redesign included in this phase

06 / PRODUCT SPINE

Phase 1 - Account + Workspace Foundation v0.1

> **Product boundary**
>
> Account answers: “Who am I and what governed workspace do I have?” Workspace answers: “Where do my sources, evidence, decisions, receipts, conversations, memory, and later skills live?” Dashboard remains the operator Gate console.

Account v0.1

- Signed-out: real sign-in, registration, verification, and provider states only if backend-supported.
- Signed-in: workspace/account name, email, workspace id when useful, plan, usage, billing state, retained receipt count.
- Primary actions: Open Workspace, Open Operator Dashboard, Sign Out.
- Compatibility: legacy account route redirects without losing query/hash callback data.
- No fake secondary workspaces if backend still models one account as one workspace.

Workspace v0.1

Surface	v0.1 behavior
Overview	Live account/workspace identity, usage, plan, receipt count, product boundary
Decisions	Counts derived only from actual retained receipts
Receipts	Actual retained receipt list; safe text rendering
Conversations	Structural surface; explicitly “persistence not connected” if no endpoint
Sources	Structural surface; no fake files/counts
Memory	Structural surface; no fabricated memories
Evidence	Structural surface initially; becomes functional in Phase 5
Settings	Session/auth state and sign out; rename/delete only if real endpoint exists

UX requirements

- Desktop: calm three-column workspace shell; mobile: usable stacked/drawer pattern with zero horizontal overflow.
- No permanent cinematic rAF loop; micro-interactions only; prefers-reduced-motion final states.
- EN / UK / RU parity; no English-only private product island.
- Private routes: noindex,nofollow,noarchive.
- No secrets in URLs; no persistent display of one-time integration API keys.

Exit gate

> **Phase 1 DONE when**
>
> A real user can authenticate, enter a real workspace context, see only backend-backed identity/usage/receipts, understand which future surfaces are not yet connected, and reach the operator dashboard without any fabricated persistence.

Phase 2 - Historical Archive + Genesis Manifest

Purpose: convert the preserved early screenshots, posters, public posts, repository metadata, and project transitions into a factual source layer before any new Genesis design is authored.

Archive structure

```
genesis/
├── archive/
│   ├── 2025-07/
│   │   ├── originals/
│   │   ├── derivatives/
│   │   └── metadata.json
│   └── ...
├── data/
│   ├── artifacts.json
│   ├── repositories.json
│   ├── lineage.json
│   ├── eras.json
│   └── milestones.json
└── tools/
    └── build-genesis-manifest.*
```

Artifact admission

State	Meaning
ARCHIVED	Preserved original; not automatically part of narrative
REVIEWED	Provenance/date/context inspected
ADMITTED	Supports a specific Genesis claim/transition
WITHHELD	Preserved but not used because relevance/provenance is insufficient

Rule: archive != admitted narrative evidence. Personal, irrelevant, duplicate, unverifiable, or contextless material can remain preserved without entering Genesis.

Historical claims policy

> **Preserve, do not endorse**
>
> Early material containing language such as “AGI signal”, “self-awareness”, or similar emergence framing should remain visible when historically relevant, but must be labeled as historical framing rather than current technical claims. The evolution of claim precision is itself part of the story.

Repository inventory

- Harvest facts: repository name, creation date, first commit date, default branch, fork flag, archived status, visibility, description, current HEAD.
- Classify: FIRST PARTY, EXPERIMENT, SUCCESSOR/DERIVED, FORK, EXTERNAL REFERENCE, ARCHIVED.
- Do not let forks increase first-party ecosystem counts.
- Generate facts automatically; curate conceptual lineage manually.

Exit gate

> **Phase 2 DONE when**
>
> The entire factual narrative can be regenerated from structured manifests, every displayed historical artifact has provenance metadata, and first-party repository evolution is separated from forks/external lineage.

08 / LIVING HISTORY

Phase 3 - Genesis v03: Evidence-Backed Evolution Trace

Genesis becomes a data-driven documentary surface: the system's history is shown through admitted artifacts and repository evolution rather than only through abstract cinematic scenes.

Narrative eras

Era	Core content	Visual principle
00 - Before Code	Earliest preserved context before repository creation	Minimal trace; no mythology
01 - The Signal	First public posts / reactions / original artifacts	Original evidence foregrounded
02 - Early Language	Early symbols/posters/framing	Historical, not current brand authority
03 - Structure	Early diagrams / structured concepts	Narrative starts becoming architecture
04 - First Repository	First first-party repo + first commit	One node, one verifiable origin
05 - Expansion	Repository count and branches grow	Data-driven graph, chronological
06 - Convergence	Experiments converge into SaC/release-control thesis	Graph collapses toward architecture
07 - Release Authority	Candidate → Gate → decision → receipt	Semantic motion, no decorative space
08 - Productization	Bridge, Local, Gate, web surfaces	Architecture becomes product
09 - Evidence	Benchmark, Book, Research	System begins inspecting itself
10 - Workspace	Private product context appears	User becomes part of system
11 - Current Head	Today's actual state	Living continuation, not "The End"

Benchmark connection

Genesis shows time and lineage. Benchmark shows current observable repository evidence. A repository node may display a compact current evidence state and deep-link into Benchmark, but Genesis must not duplicate scoring logic.

```
GENESIS = TIME / LINEAGE
BENCHMARK = CURRENT EVIDENCE
WORKSPACE = USER-OWNED CONTEXT
```

Archive old cinematic Genesis

- Preserve current cinematic build as an archive route/versioned artifact.
- Do not delete historical assets merely because v03 changes public narrative.
- Use the old cinematic build as one milestone inside the new factual Genesis: “visual language established”.

Exit gate

> **Phase 3 DONE when**
>
> Genesis is generated from structured historical/repository data, every major transition has admitted evidence, mobile/reduced-motion modes remain usable, and the current system graph is derived from curated lineage rather than random node motion.

Phase 4 - Benchmark Audit + Evidence Roadmap

The first external usage signal already points here: users understand the score, but the stronger value proposition is “what is missing and what should I improve next?”

Three-layer Benchmark product

Layer	Purpose	Output
Score	Fast hook / comparability	0-100, rank, Gate state
Audit	Explain why the score exists	Category breakdown, evidence found, evidence missing
Roadmap	Turn gaps into bounded next steps	Prioritized evidence improvements with rationale

Roadmap item schema

```
roadmap_item
├── id
├── category
├── missing_signal
├── observed_basis
├── recommended_action
├── why_it_matters
├── potential_policy_effect   # not a guaranteed score
├── confidence
├── dependencies
├── verification_method
└── status
```

Honesty constraints

- Never promise “do X and you will gain exactly N points” unless the policy deterministically guarantees it under the same snapshot conditions.
- Prefer “potential evidence gain” or “this signal would satisfy policy item Y”.
- Do not infer code quality from test file presence.
- Do not recommend security/compliance changes as if Benchmark certifies them.
- Separate repository structure evidence from executed verification evidence.

Deep indexing roadmap

Level	Capability	Status target
L1	Repository metadata + tree/path structure	Already core

Level	Capability	Status target
L2	Selected evidence documents / README / manifests	Harden
L3	Dependency/ecosystem relationships	Add after first-party lineage work
L4	Historical progression across captures	Workspace evidence history
L5	Verified execution artifacts / replay evidence	Only where safely available

Exit gate

> **Phase 4 DONE when**
>
> A user can explain the score, identify the top evidence gaps, and receive a bounded prioritized roadmap that is deterministic enough to test but humble enough not to overstate what repository evidence proves.

10 / PERSISTENCE

Phase 5 - Repository Evidence in Workspace

Benchmark becomes a source of user-owned evidence. This is the point where Workspace Evidence stops being a structural placeholder.

Canonical evidence artifact

```
workspace
└── evidence
    └── repository_snapshot
        ├── repository_identity
        ├── source_commit
        ├── captured_at
        ├── capture_mode
        ├── evidence_categories
        ├── indicators
        ├── score
        ├── gate_decision
        ├── rank
        ├── policy_version
        ├── receipt
        └── provenance
```

Required backend work

Endpoint capability	Requirement
Create evidence artifact	Authenticated; validates workspace ownership
List evidence artifacts	Paginated; stable ordering

Endpoint capability	Requirement
Read artifact	Returns immutable capture metadata
Compare captures	Optional v0.1; later for progression
Attach roadmap	Roadmap version linked to source snapshot
Delete/retention	Policy-defined; never silently mutate receipt content

Save to Workspace flow

```
BENCHMARK RESULT
   ↓
SAVE TO WORKSPACE
   ↓
AUTH CHECK
   ↓
CREATE IMMUTABLE EVIDENCE ARTIFACT
   ↓
WORKSPACE / EVIDENCE
   ↓
ROADMAP / COMPARE / CODEX TASK
```

Exit gate

> **Phase 5 DONE when**
>
> A live Benchmark result can be stored in the authenticated workspace with repository/commit/policy/receipt provenance and retrieved later without recalculating or mutating the historical result.

Phase 6 - Codex Improvement Loop v0.1

This phase turns the Benchmark roadmap into an execution loop. Codex is not asked to “make the repo better”; it receives a bounded task derived from a specific evidence gap and protected by explicit invariants.

```
SNAPSHOT A
   ↓
AUDIT GAP
   ↓
BOUNDED TASK SPEC
   ↓
CODEX IMPLEMENTATION CANDIDATE
   ↓
LOCAL TESTS + DIFF REVIEW
   ↓
COMMIT / MERGE
   ↓
SNAPSHOT B
   ↓
EVIDENCE DELTA
   ↓
RECEIPT / RESULT
```

Task template

Field	Example
Objective	Add a reproducible CI workflow for existing tests
Evidence gap	No visible CI workflow
Frozen invariants	Do not change runtime behavior or scoring policy
Files in scope	Workflow + minimal docs only
Validation	Existing tests + workflow syntax + diff check
Expected evidence effect	Satisfy current-policy CI signal if visible in captured tree
Stop condition	Any runtime behavior drift or failing tests

First improvement cases

- Add missing reproducibility guide.
- Add CI workflow around an existing test suite.
- Add deterministic fixtures/evidence artifacts where the repository already supports them.
- Document release-control decisions or receipts where architecture already exists.
- Do not fabricate functionality solely to score higher.

Anti-gaming rule

> **Benchmark score is not the goal**
>
> The Codex loop must improve observable engineering evidence only when it reflects a real engineering surface. A repository should never add meaningless files, empty tests, fake receipts, or decorative documentation purely to trigger score signals.

Exit gate

> **Phase 6 DONE when**
>
> At least one real repository completes the full loop from captured evidence gap to bounded Codex change, validation, new capture, and explainable delta - with no manual rewriting of the benchmark result.

12 / REUSABLE METHODS

Phase 7 - Skill Forge v0.1

Skill Forge converts repeated, evidenced workflows into reusable agent skill candidates. This is the bridge from “repository intelligence” to an agent skill product.

Core rule

> **Generation is not skill admission authority**
>
> Codex may detect a repeated workflow and generate a skill candidate. The candidate is not an admitted skill until it passes review, evidence checks, compatibility checks, and explicit admission. This mirrors SemeAI release-control semantics.

First case - generate the creator’s own skills

The first end-to-end Skill Forge case should use first-party SemeAI repositories because they contain the richest evidence of repeated engineering behavior. This creates a controlled environment where the creator can judge whether Codex extracted a real reusable method or merely produced plausible prose.

```
FIRST-PARTY REPOSITORIES
   ↓
COMMIT / PR / DOC / TEST PATTERNS
   ↓
CODEX WORKFLOW EXTRACTION
   ↓
SKILL CANDIDATE
   ↓
EVIDENCE REFERENCES
   ↓
HUMAN REVIEW
   ↓
```

```
ADMITTED SKILL
   ↓
USE ON A NEW REAL TASK
   ↓
REPLAY / EVALUATE
```

Candidate #1: bounded-repository-change

```
Audit current state
→ identify smallest coherent next change
→ freeze invariants
→ implement
→ run targeted + regression tests
→ inspect evidence
→ document change
→ ship only after verification
```

This candidate is strong because it reflects an already repeated working method and can be evaluated on new repositories. It should not be admitted merely because it sounds coherent.

Skill candidate schema

Field	Purpose
skill_id / name	Stable identity
version	Immutable release version
purpose	Bounded job the skill performs
inputs / outputs	Explicit contract
workflow	Steps / policy
constraints	What the skill must not do
evidence_refs	Repos/commits/docs/tests that support extraction
compatibility	Agent/runtime/tooling requirements
evaluation	How to test correct use
risk_class	Review expectations
admission_state	CANDIDATE / REVIEW / ADMITTED / RETIRED
receipt	Admission/release decision trace

Exit gate

> **Phase 7 DONE when**
>
> Codex generates at least one skill candidate from real first-party evidence; the candidate is reviewed, admitted as a versioned artifact, then successfully reused on a new real task with an evaluation record.

13 / PRODUCTIZATION OF SKILLS

Phase 8 - Skill Registry + Compatibility

Before a marketplace, skills need a trustworthy registry. The registry is the technical substrate; the marketplace is a commercial view on top of admitted registry artifacts.

Registry capabilities

Capability	v0.1 requirement
Versioning	Immutable released versions; explicit supersession
Install/export	Defined artifact format; no hidden mutation
Compatibility	Agent/runtime/tool versions and required permissions
Provenance	Creator, source evidence, generation method, admission record
Evaluation	Minimum evaluation suite / examples
Lifecycle	Candidate → Review → Admitted → Deprecated/Retired
Receipts	Admission and release decision trace retained
Dependencies	Explicit skill/tool/library dependencies
Permissions	Declared file/network/tool access expectations

Compatibility matrix

Do not market a skill as “works with all agents”. Compatibility must be explicit and versioned. Start with the agent/runtime actually used by the first internal case; expand only after verified installations.

Exit gate

> **Phase 8 DONE when**
>
> An admitted skill can be exported/installed, version-resolved, evaluated, and retired without ambiguity; a consumer can inspect provenance and requirements before use.

14 / COMMERCIAL HYPOTHESIS

Phase 9 - Optional Agent Skill Marketplace

> **Marketplace is a hypothesis, not a promise**
>
> The marketplace should launch only if admitted skills are genuinely reusable and at least a small external group wants to acquire them. The first proof is internal: creator skills generated with Codex, admitted, reused, and evaluated. Only then test external sellers and buyers.

Marketplace value proposition

- For creators: turn proven workflows into versioned, sellable agent skills rather than selling prompts with unclear provenance.
- For buyers: inspect compatibility, evidence, version history, permissions, evaluations, and admission receipts before acquisition.
- For SemeAI: use evidence and release-control as the trust layer between generation and distribution.

Possible marketplace objects

Object	Role
Listing	Commercial presentation of one admitted skill/version family
Seller profile	Identity, provenance, support/maintenance state
Skill artifact	Versioned technical package from Registry
Compatibility declaration	Supported agents/runtimes/tooling versions
Evaluation pack	Tests/examples/expected outcomes
Permissions manifest	Declared access required by the skill
Receipt bundle	Admission/release evidence
Purchase/license	Commercial right to obtain/use a version
Review signal	Consumer feedback; never release authority

First marketplace catalog

Do not start with dozens of generated skills. Start with a very small first-party catalog produced by the Codex Skill Forge loop and used repeatedly in real work. Example categories:

- Repository audit & bounded change skills.
- Evidence / release-check preparation skills.
- Documentation / reproducibility skills.
- Repository-to-roadmap skills.
- Governed skill authoring / evaluation skills.

Commercial models to test

Model	When useful	Risk
Free / open skill	Discovery and ecosystem growth	Low willingness-to-pay signal
One-time paid version	Stable bounded skill	Maintenance/version expectations
Subscription / updates	Frequently changing integrations	Churn and support load
Creator revenue share	Third-party seller ecosystem	Fraud/quality disputes
Private team skill packs	B2B internal workflows	Needs private registry/access controls
Verified/admitted badge tier	Trust surface if evidence is meaningful	Must not become pay-to-trust

Marketplace trust rules

- Payment does not grant admission, ranking, or Gate authority.
- Popular does not mean safe/correct; popularity is presentation metadata only.
- Every paid listing points to a specific admitted version, not a mutable “latest” blob without history.
- Updates are new candidates and must pass the same release/admission path.
- Seller claims must be bounded to demonstrated compatibility/evidence.
- Refund/support policy should be product policy, not encoded as model-generated promises.

Exit gate before public launch

Gate	Minimum evidence
Reusability	At least several successful uses of first-party skills on new tasks
Installability	Repeatable install/export flow
Compatibility	Explicit supported agent/runtime versions
Evaluation	Automated or reproducible evaluation pack
Trust	Provenance + admission receipt visible
Commerce	Payment/license flow tested end-to-end
Support	Clear update/deprecation path
Demand	External users ask to acquire/use at least one skill

15 / ONE PRODUCT

Phase 10 - Cross-System Integration & Final Polish

Only after the core product loop exists should SemeAI receive a final site-wide polish. Before that, large redesigns risk being invalidated by new Account/Workspace/Skill navigation and state requirements.

Cross-system transitions

From	To	Why
Home	Account / Workspace	Public interest becomes product entry
Genesis	Benchmark	Historical repo node → current evidence
Benchmark	Workspace Evidence	Public inspection → private retained artifact
Workspace Evidence	Roadmap	Stored evidence → next bounded action
Roadmap	Codex task	Recommendation → execution candidate
Codex result	Benchmark	Change → re-measure
Repeated task	Skill Forge	Execution pattern → reusable method candidate
Skill Registry	Marketplace	Admitted artifact → commercial listing

Final polish checklist

- Navigation vocabulary is consistent across public/private surfaces.
- Gold/cyan/violet meanings are stable across routes; statuses never change color arbitrarily.
- Home explains the entire loop in under 10 seconds without becoming a feature catalog.
- Genesis is documentary; Benchmark is analytical; Workspace is operational; Marketplace is commercial.
- Mobile, No-JS, reduced motion, keyboard, touch targets, and screen-width regressions are tested.
- Performance budgets and animation lifecycle are enforced.
- SEO applies to public surfaces; private product routes remain noindex.

16 / MARKET PROOF

Phase 11 - Pilot, GTM, and Monetization Evidence

The roadmap is not complete when the product works. It is complete when external users can understand, use, repeat, and eventually pay for a valuable loop.

Pilot ladder

Pilot	Goal	Evidence
A - Benchmark user	Does the user understand score/audit?	Repo submitted without hand-holding
B - Roadmap user	Does the user act on recommendations?	At least one roadmap item implemented
C - Workspace user	Does retained evidence matter?	Returns to compare snapshots/receipts
D - Codex loop user	Does measure→change→verify save time?	Repeated bounded tasks
E - Skill user	Is an admitted skill reusable?	Skill used on new work
F - Marketplace buyer	Will someone pay for a skill?	Paid acquisition + successful use

Monetization order

- Do not wait for Marketplace to monetize. Workspace plans, higher evidence retention, team features, or assisted audits may monetize earlier if useful.
- Marketplace revenue should be treated as an optional later stream, not the sole business model.
- First paid skill transaction is a market milestone only if the buyer can actually install/use the skill and receives the promised artifact/version.

Data Model and API Evolution

Recommended conceptual model

```
user
└── workspace
    ├── sources
    ├── conversations
    ├── memories
    ├── evidence_artifacts
    │   └── repository_snapshots
    ├── roadmaps
    ├── decisions
    ├── receipts
    ├── codex_runs
    ├── skill_candidates
    └── admitted_skills (references)

registry
└── skill
    └── versions
        ├── artifact
        ├── provenance
        ├── compatibility
        ├── evaluation
        └── admission_receipt

marketplace
└── listing → admitted skill version
```

API evolution by phase

Phase	New API surface
1	Reuse existing auth/account/usage/receipts; no invented endpoints
5	workspace evidence CRUD/read/list + snapshot persistence
4/5	roadmap generation/storage tied to snapshot/policy version
6	Codex task metadata/run/result references (execution may remain external initially)
7	skill candidate create/read/review/admit
8	registry version/read/install/export/evaluation metadata
9	listing/search/purchase/license/download/update state

Immutable vs mutable

Immutable / append-only	Mutable
Captured repository snapshot metadata	Workspace display name

Immutable / append-only	Mutable
Benchmark receipt / decision trace	Roadmap item status
Admitted skill version	Marketplace listing copy/pricing
Admission receipt	Seller profile
Historical Genesis original artifact checksum	Current navigation/presentation

18 / PRODUCT INTEGRITY

Governance, Trust, and Safety Boundaries

Authority map

Thing	Can propose	Can authorize
LLM output	Model/Codex	Gate / policy
Repository roadmap item	Benchmark/audit logic	User/team planning choice
Code change	Codex	Human/CI/repository workflow
Skill candidate	Codex/Skill Forge	Skill admission policy + reviewer
Marketplace listing	Seller	Marketplace publication policy
Skill popularity	Users/market	Nobody; never release authority

Privacy / security presentation

- Private workspaces and skill drafts must not be indexed publicly.
- Repository secrets, private source content, tokens, API keys, and credentials must not leak into evidence manifests, URLs, receipts, or marketplace previews.
- Skill permissions should be explicit. A buyer must know whether a skill expects filesystem, network, shell, GitHub, or other tooling access.
- Do not claim “safe skill” from a Benchmark score. Skill evaluation and admission are separate evidence paths.
- Marketplace seller verification, payment processing, and legal terms should be implemented only when required by actual commerce scope.

19 / WHAT TO MEASURE

Metrics and Evidence of Progress

Product metrics by layer

Layer	Primary metrics
Account/Workspace	sign-in success, workspace entry rate, repeat sessions
Benchmark	repo submissions, successful captures, audit expansion rate
Roadmap	roadmap viewed, item opened, item completed
Evidence persistence	saved snapshots, return-to-compare rate
Codex loop	task completion, tests passing, evidence delta, rollback rate
Skill Forge	candidates generated, admission rate, reuse success
Registry	install success, compatibility failure, update/deprecation health
Marketplace	listing conversion, paid acquisition, refund/support rate, repeat buyers

North-star candidates

> **Do not use raw score uplift as the north star**
>
> A healthier north star is “verified improvement loops completed”: a repository snapshot produces an evidence gap, a bounded change is applied, verification passes, and a new snapshot shows a meaningful observable improvement. For skills: “admitted skill successfully reused on a new task”.

20 / TESTING

Validation and Release Discipline

Every phase must end with

- Targeted tests for the new layer.
- Regression suites for frozen public/benchmark/book behavior.
- git diff --check / formatting validation.
- Mobile and desktop route/viewport checks.
- No unexpected console errors.
- Reduced motion + visibility/offscreen lifecycle where motion exists.
- Live verification after deployment, with explicit distinction between mocked authenticated tests and real credential tests.
- Structured mission report with starting state, deltas, tests, commits, deployment, invariants, weaknesses, final HEAD.

Additional Skill/Marketplace tests

Area	Required tests
Skill artifact	schema validation, deterministic package/hash where applicable
Compatibility	supported/unsupported agent/runtime versions
Evaluation	known-good, known-bad, edge-case tasks
Admission	candidate cannot self-admit; review decision retained
Update	new version does not mutate old version
Marketplace	purchase grants exact artifact/version; seller cannot swap bytes post-sale
Permissions	declared permission mismatch surfaces before use

21 / FAILURE MODES

Key Risks and Mitigations

Risk	What failure looks like	Mitigation
Feature sprawl	Many surfaces, no product loop	Dependency gates; one phase at a time
Benchmark gaming	Repos add empty evidence solely for points	Anti-gaming policy; verify meaningful artifacts
Claim drift	“Benchmark score = quality/safety”	Bounded copy + explicit evidence limits
Workspace fiction	UI shows data not persisted by backend	Structural placeholder labels until endpoints exist
Codex overreach	Generic refactors outside audit gap	Bounded task specs + frozen invariants
Skill hallucination	Plausible skill not actually derived from workflow	Evidence refs + reuse evaluation before admission
Marketplace junk	Flood of low-quality generated skills	Registry/admission required before listing
Pay-to-trust	Paid sellers gain authority/rank	Commerce metadata never Gate authority
Compatibility chaos	Skills break across agents/runtime versions	Explicit compatibility + evaluation matrix
Maintenance debt	Too many versions/listings unsupported	Deprecation/retirement lifecycle
Premature polish	Redesign invalidated by new product spine	Final polish only after integration phase

22 / GO / HOLD CRITERIA

Decision Gates

Decision	GO when	HOLD when
Build Genesis v03	Archive + lineage manifests are trustworthy	Narrative still depends on memory/guessing
Add Codex task execution	Roadmap items are bounded and verifiable	Recommendations are vague/generic
Build Skill Forge	At least one repeated workflow has strong evidence	No repeatable patterns to extract
Admit first skill	Candidate works on a new real task	Only prose quality is demonstrated
Build Marketplace	Skills are installable, versioned, reused; external demand exists	Only internal excitement; no acquisition intent
Enable paid skill sales	Purchase/license/download/support path works	Artifact delivery/version semantics unresolved
Final polish	Core IA and loops stable	Major product layers still changing

23 / IMMEDIATE EXECUTION

Next Six Codex Missions

Mission 1 - Account & Workspace Foundation v0.1

Ship real private product spine using existing backend; no fake persistence.

Mission 2 - Genesis Historical Archive & Evidence Manifest

Ingest preserved originals, checksum/provenance, repository inventory, fork classification, curated lineage; no redesign.

Mission 3 - Genesis v03: Evolution Trace

Render documentary data-driven history; archive cinematic v02.

Mission 4 - Benchmark Audit + Evidence Roadmap

Explain score, surface evidence gaps, map bounded next actions; test on external repos.

Mission 5 - Workspace Repository Evidence

Persist Benchmark snapshot/receipt/roadmap as workspace evidence.

Mission 6 - Codex Improvement Loop v0.1

Execute one bounded roadmap task, verify, recapture, compare delta.

After Mission 6, stop and evaluate whether the loop is genuinely useful before starting Skill Forge. Skill Forge is Phase 7, not a shortcut around the evidence loop.

Marketplace Blueprint - If the Hypothesis Is Validated

Buyer journey

```
DISCOVER
  ↓
INSPECT LISTING
  ↓
CHECK COMPATIBILITY + PERMISSIONS + EVIDENCE
  ↓
ACQUIRE / LICENSE
  ↓
INSTALL EXACT VERSION
  ↓
RUN EVALUATION / FIRST USE
  ↓
RECEIPT / SUPPORT / UPDATE PATH
```

Seller journey

```
WORKFLOW EVIDENCE
  ↓
SKILL CANDIDATE
  ↓
REVIEW + EVALUATION
  ↓
ADMITTED VERSION
  ↓
CREATE LISTING
  ↓
PRICE / LICENSE / SUPPORT
  ↓
PUBLISH
  ↓
NEW UPDATE = NEW CANDIDATE
```

SemeAI's differentiation if built well

- Not "a prompt store": the sellable object is a versioned skill artifact with explicit provenance and evaluation.
- Not "AI generated = trusted": generated skills remain candidates until admission.
- Repository evidence can show where a skill came from; receipts can show which version/admission decision was published.
- The same workspace can contain the repository evidence, improvement history, skill candidate, evaluations, and admitted release.

What NOT to build in marketplace v0.1

- Complex recommendation algorithms.

- Token economies or speculative incentives.
- Hundreds of autogenerated listings.
- Universal agent compatibility badges.
- Paid ranking that appears to be technical trust.
- Automatic publication immediately after Codex generation.

Target End State

> **The complete loop**
>
> A user discovers SemeAI, understands its history and release-control thesis, enters a real workspace, inspects a repository, saves a deterministic evidence snapshot, receives a bounded improvement roadmap, applies a Codex task, verifies the result, and can later turn a repeated method into an admitted reusable skill. If external demand appears, admitted skills can become marketplace products.

```
DISCOVER
   ↓
UNDERSTAND (Genesis / Book / Gate)
   ↓
INSPECT (Benchmark)
   ↓
OWN CONTEXT (Workspace)
   ↓
IMPROVE (Roadmap + Codex)
   ↓
VERIFY (new snapshot + receipt)
   ↓
REUSE (Skill Forge)
   ↓
DISTRIBUTE (Registry)
   ↓
SELL / ACQUIRE (Marketplace, if validated)
```

Final strategic principle

Do not optimize SemeAI for the number of features. Optimize it for the integrity of the loop from observable evidence to bounded action to retained decision trace. The marketplace is valuable only if that integrity survives commercialization.

Appendix A - Phase Exit Checklist

Phase	Exit checklist
Phase 1	Real auth/workspace; no fabricated surfaces; private routes noindex; responsive/i18n/tests pass.
Phase 2	Original artifacts checksummed; provenance recorded; first-party/fork classification complete; lineage curated.
Phase 3	Genesis v03 data-driven; historical claims labeled; old cinematic version archived; benchmark not duplicated.
Phase 4	Audit explains score; roadmap bounded; no guaranteed uplift claims; external repo tests completed.
Phase 5	Snapshot+commit+policy+receipt persisted immutably; workspace ownership enforced.
Phase 6	One real measure→change→verify loop completed; anti-gaming rule respected.
Phase 7	Codex-generated creator skill reviewed, admitted, and reused on a new task.
Phase 8	Skill versioning/install/compatibility/evaluation/deprecation proven.
Phase 9	External demand + commerce + exact artifact delivery proven before “marketplace launch”.
Phase 10	Cross-system IA stable; final polish no longer changes architecture.

Appendix B - Suggested Repository/Skill Status Vocabulary

Domain	States
Benchmark presentation	SHOW / REVIEW / BLOCK
Internal release control	PROCEED / NEEDS_REVIEW / SILENCE
Historical artifact	ARCHIVED / REVIEWED / ADMITTED / WITHHELD
Roadmap item	PROPOSED / ACCEPTED / IN_PROGRESS / VERIFIED / SKIPPED
Skill lifecycle	CANDIDATE / REVIEW / ADMITTED / DEPRECATED / RETIRED
Marketplace listing	DRAFT / PUBLISHED / PAUSED / DELISTED

Appendix C - Marketplace Readiness Questions

- Can a buyer inspect exactly which skill version they will receive?
- Can the same skill version be reproduced or re-downloaded without silent mutation?
- Does the skill declare agent/runtime compatibility and required permissions?
- Is the skill supported by real usage evidence rather than only generated documentation?
- Can an update be reviewed as a new candidate instead of mutating the sold version?
- Can a seller make bounded claims without implying safety/certification?
- Can payment fail without affecting technical admission/Gate decisions?
- Can a buyer evaluate the skill before high-stakes use?
- Is there a deprecation/support path?
- Is there real external demand strong enough to justify marketplace complexity?