# ATO Bot

## Technical White Paper

Governed full-stack NIST 800-53 assessment with evidence preparation, code-bounded adjudication, and human-reviewable remediation

July 6, 2026

**Document Purpose**

This white paper describes ATO Bot as a technical platform rather than a product pitch. It focuses on how the application prepares evidence, executes full control-level assessment, adjudicates findings through deterministic policy logic, preserves human authority, and carries the result through remediation and reporting.

### Core Narrative



### Appendices

# Contents

# 1. Executive Summary

ATO Bot is a full NIST 800-53 assessment platform built around a core design choice: the system does not treat uploaded documentation, model output, or generated narrative as authoritative in isolation. Instead, it assembles assessment-ready evidence, routes that evidence through bounded reasoning tasks, applies deterministic adjudication policy, and keeps the reviewer in authority over what becomes an assessment result.

The result is a platform that sits between an evidence-preparation subsystem, a governed assessment engine, a reviewer workbench, and a post-assessment remediation and reporting system. Ingestion matters because it prepares evidence for the assessment engine, but the center of gravity of the product is full control-level assessment, code-based adjudication, human review, remediation, and export.

- Evidence enters through project, enterprise, and common-control libraries so the assessment engine can evaluate both system-specific and inherited control context.
- Parsing, screening, expansion, classification, and embedding create reusable evidence units rather than ephemeral prompt context.
- Assessment reasoning is model-assisted, but final control status is governed by objective logic, adjudication policy, and human review workflows.
- Dissent handling, remediation guidance, artifact generation, reporting, and export are integrated into the same assessment system rather than bolted on afterward.
- The architecture is strongest where it performs governed full assessment and then operationalizes the result through closure, remediation, and formal outputs.

**Technical Thesis**

ATO Bot should be understood as a governed full-assessment system for federal authorization work, not as an ingestion utility or generic AI document generator.

# 2. Problem Statement and Design Goals

Traditional ATO preparation is fragmented. Evidence lives across project folders, enterprise policy repositories, spreadsheets, screenshots, system exports, and tribal knowledge. Assessment work is correspondingly repetitive: teams re-assemble context, re-evaluate the same objectives, make inconsistent adjudication calls, and re-create reviewer narratives from scratch for every package or reassessment cycle.

Many AI approaches fail here because they collapse the workflow into a single prompt: upload documents, ask for a verdict, and trust the narrative. That pattern is fast to demo but weak under audit, because it obscures provenance, blurs assessment logic with prose generation, makes retrieval behavior difficult to inspect, and leaves little room for controlled human adjudication.

- Preserve provenance from raw upload to final assessment statement.
- Separate model-assisted reasoning from final policy and control-status decisions.
- Standardize adjudication behavior across controls through explicit assessment policy rather than per-run scoring drift.
- Keep human review, challenge, override, and export as first-class product behaviors.

- Allow evidence to be reused across projects, common controls, enterprise libraries, and later reporting surfaces.
- Create a product architecture that can carry a finding from evidence preparation through assessment, dissent, remediation, reassessment, and formal reporting.

**Design Goal**

The system is optimized for reliability, repeatability, and trust: a reviewer should be able to see what evidence was used, how the control was adjudicated, why the system leaned a certain way, and what would be needed to close any remaining gap.

# 3. Product Scope and Operating Model

ATO Bot spans the full operating path from evidence preparation through full control assessment, reviewer challenge, remediation, and formal outputs. The current application surface includes project management, multi-scope document libraries, ingestion and retrieval, control assessment, reviewer workbenches, remediation and closure, reporting and export, AI assistant workflows, system knowledge extraction, integrations, and administrative runtime management.

- Project and assessment workspaces for the live system boundary under review.
- Project document library plus enterprise policy, enterprise procedure, and common-control provider libraries.
- Assessment policy surfaces for bucket logic, overrides, and rollout governance.
- Assessment review, dissent handling, evidence traceability, and control-level decision inspection.
- Remediation guides, remediation artifact generation, POA&M-related outputs, and report generation.
- SSP Workbench, OSCAL export runs, and reporting endpoints.
- System Knowledge, Architecture and Tools, optional integrations, and operational-observability scaffolding.
- Prompt manager, AI runtime configuration, user/security admin, audit logging, and synthetic dataset generation.

Operationally, the platform behaves like a governed assessment system built on top of a staged evidence substrate. Documents do not simply sit in storage; they are parsed, promoted, classified, embedded, assembled into objective packets, adjudicated into findings, challenged, remediated, and eventually exported into formal assessment and reporting artifacts.

**Current product truth**

The flagship path today is governed full assessment review. Ingestion is the preparation layer for that engine, while operational observability helps operators understand application health, job state, audit activity, configuration changes, and processing status. Optional security-posture integrations are disabled by default and are outside the supported assessment claim.

# 4. Platform Architecture

The runtime architecture combines a React single-page application, a FastAPI application server, a background worker, PostgreSQL with pgvector, Redis, file-based upload and output storage, and

multiple model/provider routes. This separation is important because ATO Bot performs long-running ingestion and assessment work that should not be tightly coupled to browser session behavior.

Figure 1 shows the dominant runtime pattern. The browser and frontend handle workspace interaction. The backend owns orchestration, business rules, API composition, and persistence. The worker executes ingestion, assessment, remediation, and package-generation jobs. PostgreSQL stores both structured workflow state and semantic embeddings, while Redis supports coordination and runtime support functions.

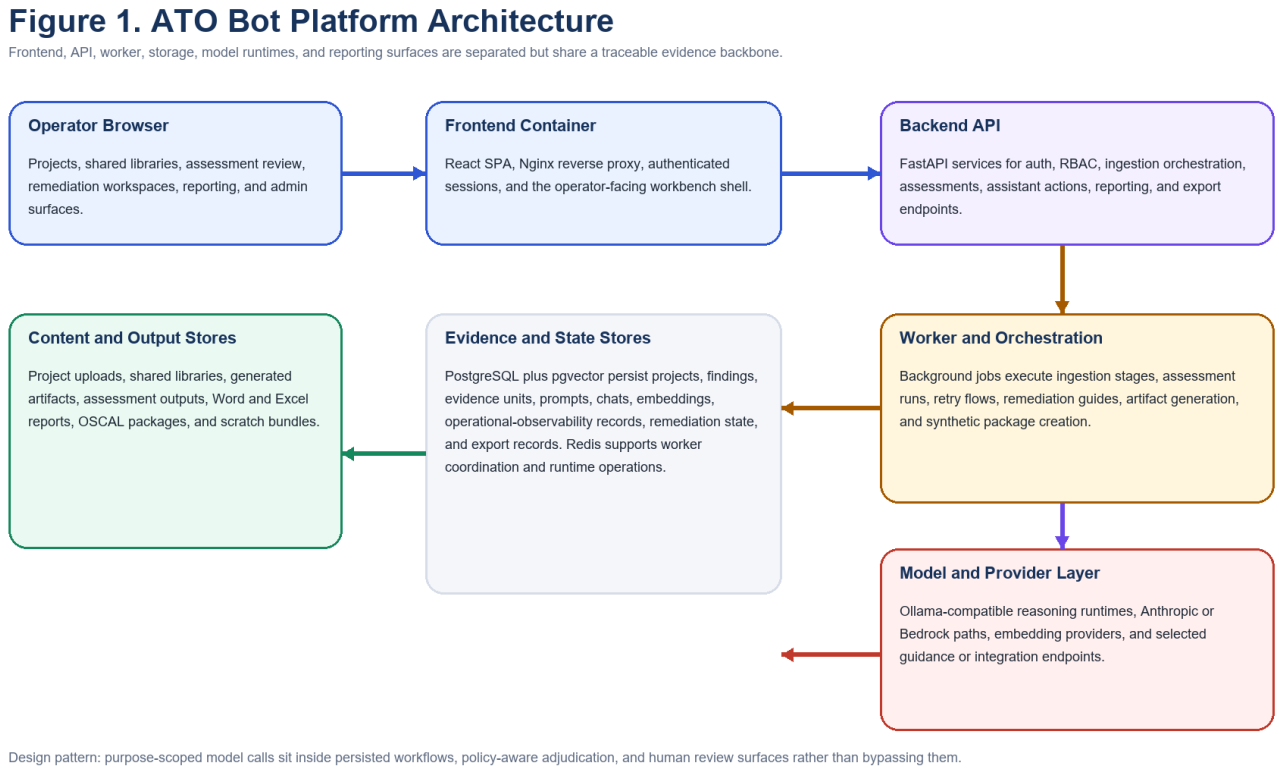



Figure 1. ATO Bot platform architecture.

## Architecture characteristics

- The frontend is a React 18, Vite, Tailwind, React Router, and React Query application behind Nginx.
- The backend is a FastAPI and SQLAlchemy asyncio stack with Alembic migrations, JWT auth, MFA support, rate limiting, and role-aware APIs.
- The data layer uses PostgreSQL for workflow state and pgvector for retrieval over evidence embeddings.
- The worker model allows ingestion, assessment, remediation, and synthetic test jobs to run asynchronously from the operator interface.
- Uploaded and generated artifacts are persisted on disk, creating a practical document store for both inputs and outputs.

This architecture gives ATO Bot a stronger trust posture than systems that rely on browser-only prompt flows. It preserves state, exposes explicit APIs, and makes room for persistent governance objects such as assessment policy, overrides, rollups, findings, remediation reports, and export runs.

# 5. Evidence Ingestion and Normalization Pipeline

ATO Bot's ingestion model is the evidence-preparation subsystem for the assessment engine. Its job is not to act like a standalone document product. Its job is to prepare source artifacts for control-level 800-53 evaluation through an exhaustive, staged normalization path that preserves provenance and improves later reasoning quality. The platform does not treat a document as a blob that is only unpacked at prompt time. It turns uploads into canonical source records, screens them for relevance, expands them into reusable evidence units, classifies their role and strength, embeds them for retrieval, and preserves compatibility state needed by the current assessment stack.

Figure 2 summarizes the evidence-preparation path from upload to assessment-ready state. Each stage produces durable intermediate objects so the assessment engine can recover from failures, inspect quality, explain what happened, and reuse evidence across later assessment, remediation, assistant, and export flows.

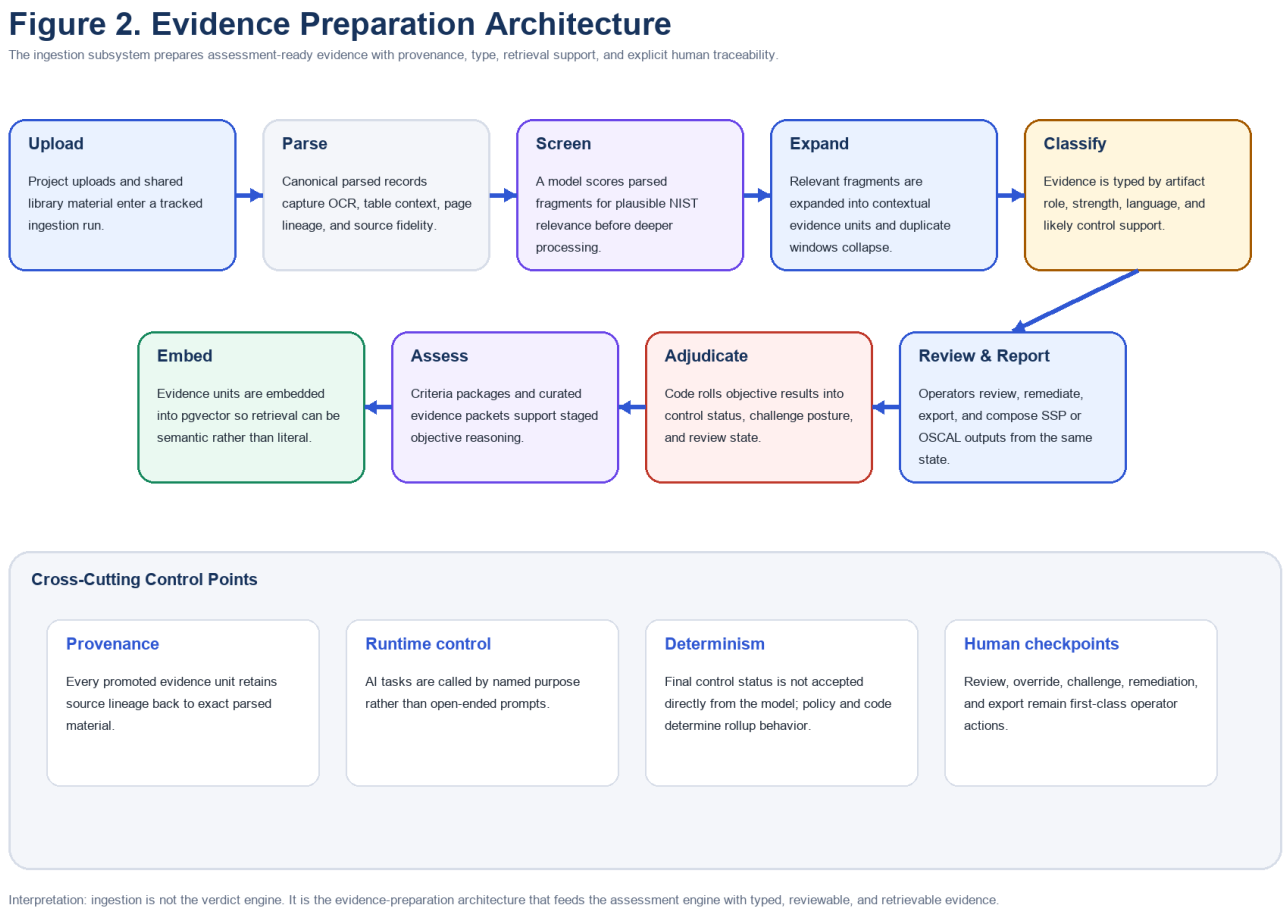



Figure 2. Evidence preparation architecture.

## Library entry points

- Project document library for system-specific evidence.
- Common-control provider library for inherited or shared evidence domains.
- Enterprise policy library for organization-wide governance materials.
- Enterprise procedure library for reusable implementation procedures and operational records.

Those entry points matter because ATO Bot prepares evidence across multiple responsibility layers before assessment begins. Project evidence captures system-specific implementation. Enterprise and common-control libraries provide reusable governance and inherited-control context. The ingestion pipeline is designed to normalize all of those sources into a form that can be assessed together without losing their origin.

## Admission and run-state management

- Validate extension, size, and duplicate status within the target library scope before any deeper processing begins.
- Compute a content hash and create a durable document record tied to the project or shared library where the evidence entered.
- Launch a tracked ingestion run with stage status so the system can show where a document is in the pipeline and whether a failure happened at parse, screen, expand, classify, or embed time.

## Stage 1: canonical parsing and provenance capture

Parsing is the traceability foundation for the whole platform. PDFs, DOCX, XLSX, PPTX, Visio files, text, markdown, and image artifacts are converted into canonical parsed records that preserve as much source structure as possible. The system captures line numbers, pages or sheets, section paths, row and column positions, table context, OCR-derived text where needed, and enough block structure to later explain exactly where a piece of evidence came from.

- PDF handling preserves page-level provenance and can fall back to OCR for text-poor pages.
- Word parsing walks paragraphs and tables in source order so headings, procedural text, and tabular records remain distinguishable.
- Spreadsheet parsing carries header inheritance and cell coordinates so one value can be interpreted with the row and column meaning around it.
- PowerPoint parsing treats slides as page-like units and preserves title or shape context where possible.
- Image ingestion converts screenshots or scans into searchable text while still retaining the source artifact as the evidence root.

## Stage 2: model-driven screening

After parsing, the system does not immediately promote every line into assessment context. A reasoning model screens parsed units for plausible control relevance against the active NIST corpus. This is not a compliance decision. It is a recall-oriented triage step that decides which source fragments deserve deeper handling.

- The screener evaluates lines, cells, and local context rather than relying only on literal control keywords.
- It returns relevance scores, candidate controls or enhancements, and rationale that can be inspected later.
- Thresholds, batching, timeout behavior, and fallbacks make the stage operationally tunable instead of prompt-only.
- Inclusive screening reduces the chance that useful but indirectly worded evidence is dropped too early.

## Stage 3: context expansion and duplicate collapse

A triggering line on its own is often too thin to assess. The next stage expands promoted fragments into contextual evidence units by walking the same parser block, table row, section neighborhood, or local fallback window. This produces evidence that is interpretable by both humans and downstream models.

- Promotion from atomic text to contextual evidence preserves surrounding implementation detail, approvals, headers, and operational meaning.
- Duplicate collapse removes redundant expansions when several trigger lines resolve to the same effective evidence block.
- Separate corroborating evidence from different documents is preserved, which helps later confidence and contradiction handling.

## Stage 4: evidence classification

Each evidence unit is then classified so the rest of the platform knows what kind of material it is looking at. This is a major preparation step for assessment because the system can behave differently when it knows whether a record is policy, procedure, technical implementation, screenshot evidence, management approval, or testing evidence.

- Classification attaches likely control support, enhancement relevance, artifact type, evidence language, and confidence.
- Evidence strength and language type inform later retrieval, triage, and weighting behavior.
- The classification layer helps the system distinguish governance text from operational proof and concrete proof from generic narrative.

## Stage 5: semantic embedding and retrieval preparation

Once evidence is typed and contextualized, embeddings are written so assessment and assistant flows can retrieve material semantically rather than only by exact term match. This is one of the steps that makes the assessment engine reusable at scale because the same evidence substrate can support findings review, assistant workflows, report generation, and later reassessment.

- Embeddings are stored in pgvector alongside structured workflow state.
- Evidence-unit-first retrieval gives later stages a more assessment-ready corpus than raw document search would provide.
- Semantic retrieval improves recall when a document proves a control without ever naming that control explicitly.

## Compatibility, backfill, and assessment readiness

The platform still maintains compatibility with older chunk-based retrieval paths and control-tag backfills where parts of the current stack depend on them. That transitional layer is important to call out because it preserves continuity while the broader architecture keeps moving toward evidence-unit-first assessment.

- Legacy chunk and control-tag compatibility keeps older retrieval and supporting flows functional during the transition.

- Assessment scope assembly can now draw from project artifacts, shared provider evidence, enterprise libraries, and semantically prepared evidence units in one staged system.
- The final product of ingestion is not just indexed text; it is an assessment-ready evidence substrate with provenance, context, type, strength, and retrieval support.

**Why this matters**

ATO Bot does not begin assessment from scratch every time a reviewer opens a control. It begins from a staged, typed, and retrievable evidence substrate that has already been normalized, de-duplicated, classified, and prepared for control-level evaluation.

# 6. Assessment and Adjudication Engine

ATO Bot's center of gravity is the assessment engine. The platform does not stop at staging evidence; it turns that evidence into control-level NIST 800-53 assessment outcomes through a persisted pipeline that combines criteria assembly, evidence curation, objective reasoning, deterministic adjudication, challenge handling, dissent preservation, and human review.

Figure 3 shows the dominant assessment architecture. Every major assessment step has a narrow authority boundary: the model reasons over objectives and curated evidence packets, the policy system and service layer compute final control status, and the reviewer remains the final authority on what moves forward into closure, remediation, or export.

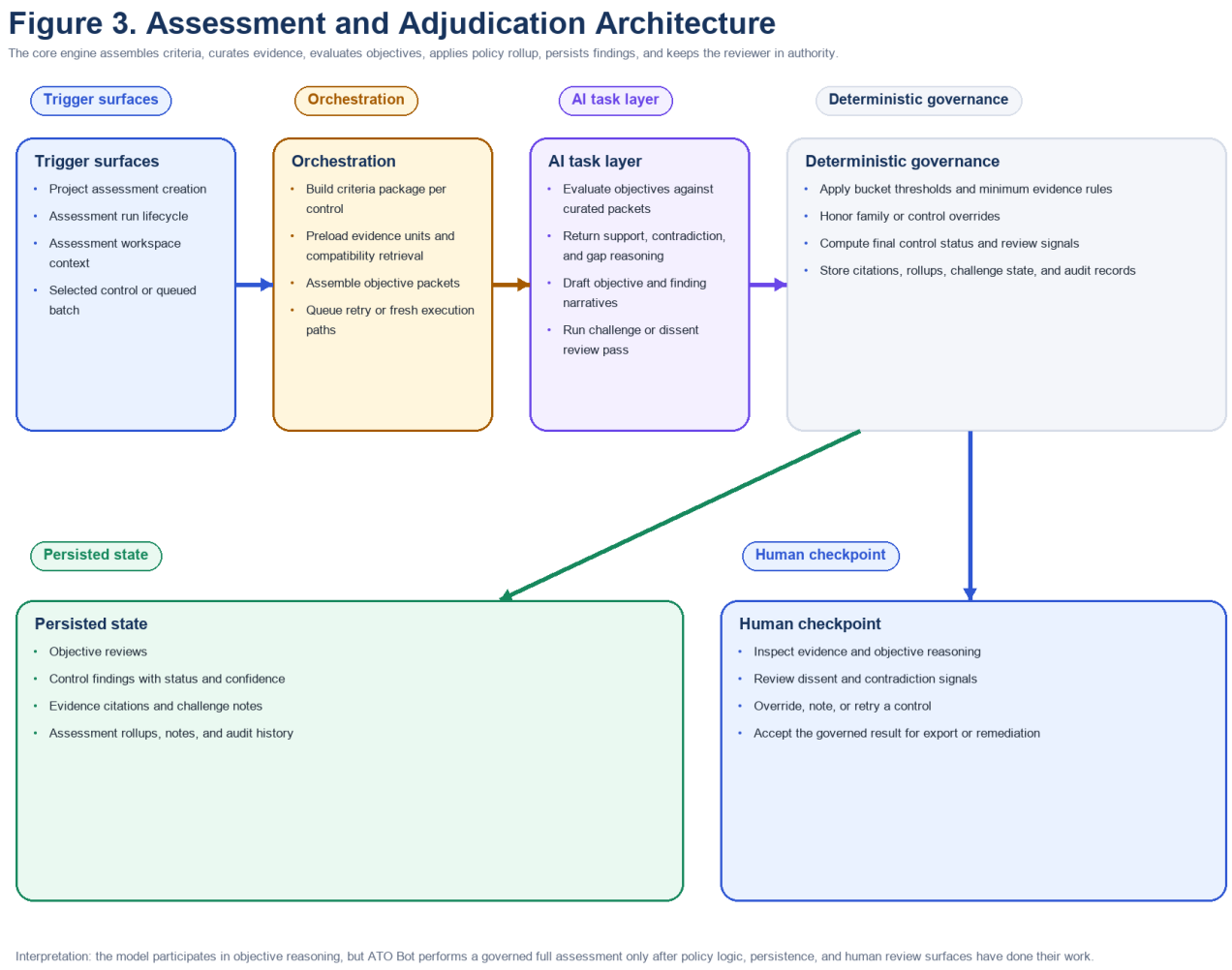



Figure 3. Assessment and adjudication architecture.

## Assessment entry surfaces and run lifecycle

Assessments are launched from the project workspace, where a reviewer creates the run, chooses the evidence scope, and later returns to the same assessment shell to review status, open findings, inspect evidence, and export results. The run itself is a first-class product object with queue state, progress state, retry history, notes, findings, dissent records, rollups, and output surfaces. That matters because ATO Bot is not performing an isolated prompt call per control. It is managing a persisted assessment lifecycle.

- Runtime purpose: execute a full control-level 800-53 assessment against a bounded evidence universe and preserve it as an accountable assessment record.
- Entry points: project assessment creation, rerun and retry actions, findings workspace navigation, and follow-on reporting or remediation flows.
- Staged flow: create assessment record, assemble criteria and evidence context, evaluate objectives, compute control status, persist findings, and expose review surfaces.
- Operational inputs: project evidence units, shared libraries, active assessment policy, baseline scope, and runtime routing metadata.

- Human checkpoint: assessors can inspect progress, challenge results, annotate findings, and decide what moves forward into remediation and export.

## Criteria package assembly

Each control pass begins with deterministic criteria package assembly. ATO Bot resolves the target control, its objective structure, assessment-policy linkage, family-specific handling, and any control-level overrides into one executable package. This package becomes the governing contract for later model reasoning and policy rollup. In practice, that means the platform never asks the model to infer what the control is supposed to mean from raw prose alone; it gives the model a structured objective target and keeps the authoritative interpretation of policy in code.

- Runtime purpose: convert the selected control into an executable assessment contract with explicit objectives and policy linkage.
- Entry points: initial assessment run, targeted retry, and control reevaluation after reviewer action.
- Operational inputs: control metadata, objective definitions, assessment policy buckets, family overrides, and control overrides.
- Model task and returned product object: none at this stage; the product object is a deterministic criteria package.
- Deterministic logic and authority boundary: code decides objective structure, override precedence, and policy linkage before any model reasoning occurs.
- Persisted state: later objective reviews and control findings remain traceable to the criteria and policy configuration in force for that run.
- Downstream product effect: all later reasoning and adjudication are constrained by the same package, which strengthens repeatability.

## Evidence assembly and assessment scope

The engine then assembles a control-specific evidence packet. It can preload semantically prepared evidence units, consult compatibility chunk paths, reuse full-document support signals, and draw from project, policy, procedure, and common-control domains. This is where the upstream ingestion system becomes subordinate to the assessment engine: ingestion prepares the substrate, but the assessment layer chooses the bounded evidence scope that a control objective is actually allowed to see.

- Runtime purpose: build the best available evidence packet for the control from the staged assessment-ready substrate.
- Entry points: initial control evaluation, retry reassessment, and reviewer evidence inspection.
- Operational inputs: evidence units, chunk compatibility paths, vector retrieval support, full-document tags, and domain metadata.
- Model task and returned product object: the model does not decide the scope; it later receives the curated packet that code has assembled.
- Deterministic logic and authority boundary: retrieval, preload limits, domain inclusion, and packet construction remain service-layer responsibilities.
- Persisted state: evidence citations and packet lineage become part of the control finding and evidence-inspection surfaces.
- Human review point: reviewers can inspect exactly what was cited, what was missing, and whether contradictory evidence was present.

## Assessment objective evaluation

Objective evaluation is the first major model-governed step. For each objective, ATO Bot sends the objective contract, the curated evidence set, and the immediate control context. The model is asked to return structured support, contradiction, gap, and narrative reasoning for that objective. It is not asked to compute the final control status. The point of the model layer here is bounded evidentiary reasoning, not policy authority.

- Runtime purpose: evaluate one objective against the evidence packet and return structured support and gap reasoning.
- Entry points: primary assessment, retry reassessment, and secondary review or challenge flows.
- Operational inputs: objective text, curated evidence excerpts, provenance, local system context, and assessment instructions.
- Model task and returned product object: objective review with support judgments, missing evidence explanation, contradiction signals, and draft narrative reasoning.
- Deterministic logic and authority boundary: schemas, parsing, evidence linkage, and downstream use of the result are code-controlled.
- Persisted state: objective reviews are stored as durable assessment artifacts, not transient assistant output.
- Downstream product effect: later control findings can cite objective-level support and gaps rather than presenting only a top-line score.

## Assessment challenge review and evidence-quality handling

ATO Bot is designed to preserve ambiguity instead of hiding it. Weak evidence, contradictory evidence, incomplete corroboration, compensating evidence, and reviewer-attention signals are all represented explicitly. This matters because many real assessments fail not on whether a document exists, but on whether the evidence record is strong enough, current enough, and internally consistent enough to justify a clean determination. The engine therefore surfaces evidence quality as a first-class property of the result.

- Runtime purpose: identify when evidence is weak, incomplete, contradictory, stale, or only partially aligned to the objective.
- Entry points: objective evaluation, challenge review, AI dissent handling, and evidence-inspection workflows.
- Operational inputs: objective review output, citations, corroboration patterns, challenge notes, and policy handling rules.
- Model task and returned product object: structured contradiction and gap language plus explanations for why evidence did not fully close the objective.
- Deterministic logic and authority boundary: evidence-quality signals, review-needed flags, and policy consequences are applied by code.
- Persisted state: gap statements, reviewer-attention flags, contradiction markers, and dissent metadata remain part of the finding record.
- Human review point: the reviewer can see exactly where the system found friction and decide whether to accept or challenge it.

## Assessment narrative generation

The assessment engine also drafts the explanatory layer that operators actually read. Objective reasoning, cited evidence, missing-proof language, and control-level explanation are assembled into a persisted finding object with status, confidence, evidence references, and remediation context. This narrative layer is important because it makes the assessment legible: the reviewer can see what the system believed, what it cited, and why the control landed where it did.

- Runtime purpose: turn structured objective results into readable control-level finding content.
- Entry points: control assessment pass, retry rerun, and downstream review or report generation.
- Operational inputs: objective reviews, evidence citations, contradiction markers, policy context, and control metadata.
- Model task and returned product object: objective narratives, control reasoning summaries, gap statements, and draft finding prose.
- Deterministic logic and authority boundary: finding schemas, confidence handling, evidence linkage, and persistence rules remain service-controlled.
- Persisted state: control findings with determination, confidence, gaps, remediation context, citations, and reviewer-facing explanation.
- Downstream product effect: the same finding feeds the findings workspace, dissent review, remediation guidance, and reporting outputs.

## Code-governed adjudication handoff

After reasoning comes adjudication, and this is where the platform most clearly separates itself from a generic AI document reviewer. ATO Bot includes an explicit assessment policy layer with threshold buckets, objective handling rules, family overrides, and control overrides. The structured objective and finding data produced upstream are fed into that deterministic layer, which computes the final control status. In other words, the model can explain what the evidence appears to show, but code still decides how that evidence maps to compliant, partial, non-compliant, inherited, or other handled states.

- Runtime purpose: convert objective-level reasoning into a policy-governed control determination that is stable and explainable.
- Entry points: every completed control evaluation, targeted retry, challenge rerun, and later rollup or export operation.
- Operational inputs: objective reviews, support and contradiction signals, confidence markers, policy buckets, family rules, and control overrides.
- Model task and returned product object: none at this stage; adjudication is computed by code from persisted structured assessment data.
- Deterministic logic and authority boundary: thresholds, override precedence, objective handling, and final status computation are the authoritative decision layer.
- Persisted state: final determination, rollup contribution, review-needed flags, dissent references, and related audit records.
- Why this matters: the system can reproduce why a control became partial or non-compliant without pretending the model alone made that decision.

## Retry reassessment and recovery behavior

Assessment runs are operational systems, which means they must recover from weak passes, incomplete packets, and transient failures. ATO Bot therefore supports bounded retry and reassessment behavior. Rather than treating the first inference pass as sacred, the orchestration layer can rebuild context, reassemble evidence, and rerun the same narrow assessment contract while preserving the fact that recovery occurred.

- Runtime purpose: recover from incomplete, weak, or failed assessment passes without discarding run history.
- Entry points: worker retry eligibility, operator retry actions, and targeted control reassessment paths.
- Operational inputs: prior findings, attempt metadata, rebuilt evidence packets, and the same criteria package.
- Model task and returned product object: a fresh structured assessment result produced under the same bounded evaluation contract.
- Deterministic logic and authority boundary: retry eligibility, attempt counting, queueing, recovery status, and persistence remain orchestration responsibilities.
- Persisted state: retries remain visible through run records, refreshed findings, and activity history rather than silently replacing older results.
- Human review point: assessors can see that recovery occurred and decide whether the rerun meaningfully resolved the original issue.

## Persisted assessment state and human checkpoints

The end state of an assessment is not just a score. ATO Bot persists objective reviews, control findings, evidence citations, dissent metadata, rollups, notes, and audit records so a reviewer can inspect the assessment as an accountable work product. This persistence-first design is what allows the same assessment to power immediate triage, remediation planning, executive reporting, export packaging, and later reassessment without losing its evidentiary lineage.

- Persisted state: objective reviews, control findings, citations, remediation seeds, dissent records, notes, activity logs, and assessment rollups.
- Human checkpoint: reviewers can inspect, challenge, annotate, override, or carry findings forward into remediation and reporting.
- Downstream product effect: findings, evidence, dissent, remediation, and export surfaces all consume the same canonical assessment state.
- Reliability and trust outcome: every important step leaves a durable object that can be audited instead of requiring trust in a transient model response.

## Assessment outputs

By the time a control leaves the assessment engine, ATO Bot has produced much more than a pass or fail label. It has produced an inspectable assessment object that can drive reviewer triage, explainability, remediation planning, and downstream reporting without reconstructing the reasoning from scratch.

- Objective determinations and their evidence review state.

- Control findings with status, confidence, gaps, remediation context, and citations.
- Assessment rollup records and ATO support rollup summaries.
- Dissent and challenge artifacts that preserve disagreement instead of hiding it.
- Reviewer notes, overrides, and activity logs that keep human intervention visible.

**Why this matters**

ATO Bot performs full 800-53 assessment by combining criteria assembly, evidence curation, objective reasoning, deterministic adjudication, persisted findings, and human authority in one governed pipeline. Ingestion improves the substrate, but the assessment engine is where the product earns reliability, repeatability, explainability, and trust.

# 7. Human Review, Remediation, and Reporting

The assessment engine produces the core result, but the product becomes operational in the review, closure, remediation, and reporting layers. ATO Bot is not only a scoring system. It is a post-assessment workbench that helps a reviewer move from assessed control state to challengeable findings, closure-ready guidance, owner-facing artifact drafts, and final exportable reporting outputs.

Figure 4 shows the review and collaboration architecture, where the same persisted assessment state is opened for findings inspection, evidence drill-down, AI dissent handling, reviewer notes, and override decisions. Figure 5 then shows the remediation and artifact-generation architecture that turns unresolved controls into guided closure work and reportable output.

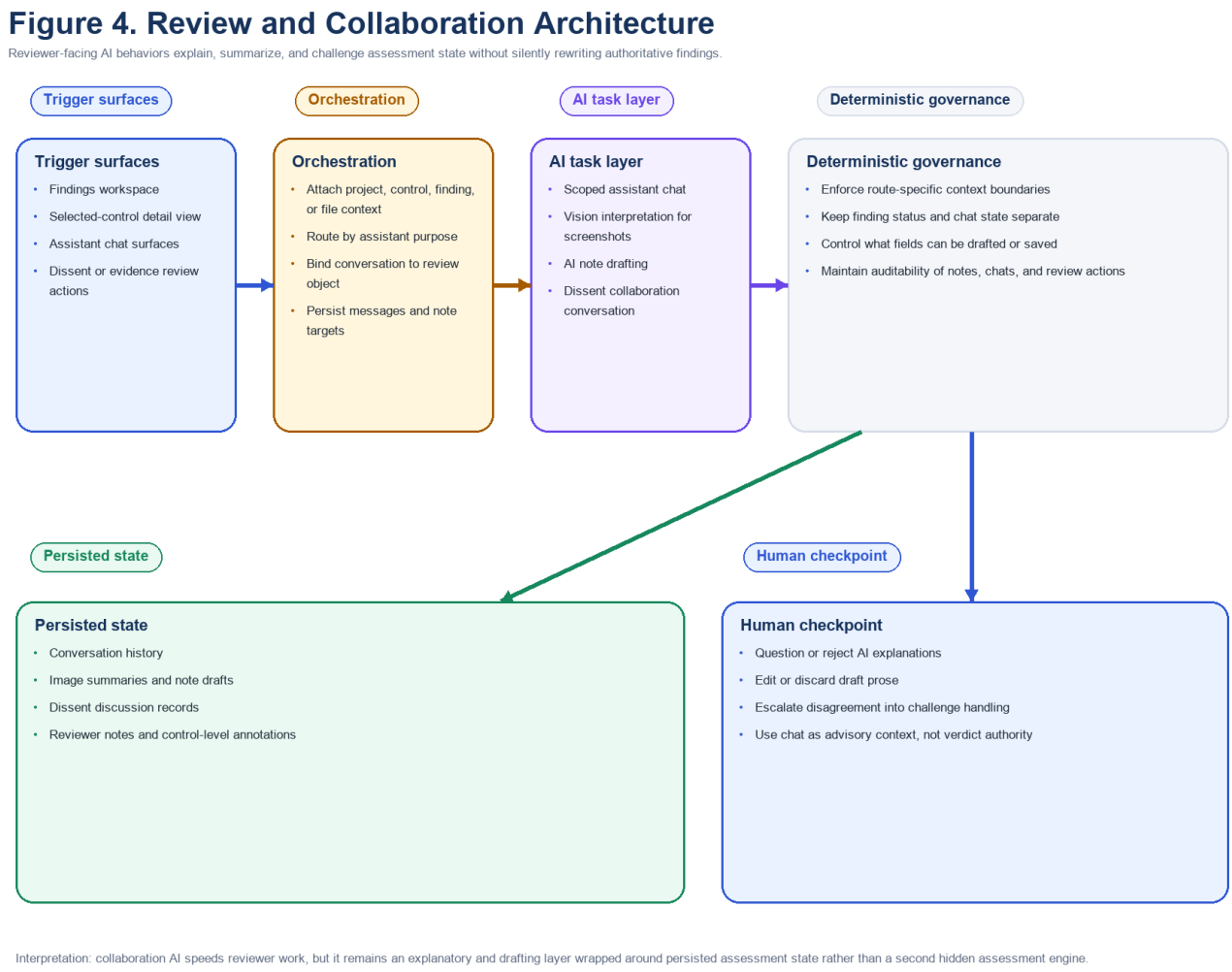



Figure 4. Review and collaboration architecture.

## Findings review workspace

The findings workspace is the primary operator surface for turning machine-produced assessment results into human action. It exposes control posture by family or flat list, allows filtering by determination and review signal, links directly into evidence and selected-control detail, and keeps the assessment-wide record visible while the reviewer works one control at a time.

- Runtime purpose: let the reviewer navigate assessment-wide results, identify the controls that need attention, and enter control-level review without losing assessment context.
- Entry points: assessment overview, findings tabs, status filters, review-needed chips, passed-control views, and selected-control actions.
- Operational inputs: persisted control findings, rollups, evidence counts, dissent flags, reviewer-attention markers, and notes.
- Model task and returned product object: none required for the workspace itself; it is a deterministic UI over persisted assessment state.
- Deterministic logic and authority boundary: filters, grouping, counts, and navigation state are application-controlled rather than inferred by the model.

- Persisted state: reviewer navigation does not rewrite findings silently; it opens the same control and assessment records already produced by the engine.
- Human review point: the operator chooses which finding to inspect, what deserves challenge, and what should move into remediation or closure.

## Evidence inspection flows

The evidence view answers the most important review question after a control is scored: what did the system actually use, what was missing, and how did it interpret the material? The reviewer can inspect selected citations, source excerpts, evidence posture, and control-specific context without leaving the assessment. This prevents the finding from becoming a black box.

- Runtime purpose: expose cited evidence, missing support, contradiction patterns, and interpretation context for one assessed control.
- Entry points: selected-control review, evidence tab navigation, citation drill-down, and slide-over control detail surfaces.
- Operational inputs: persisted citations, evidence-unit metadata, source excerpts, control reasoning, and challenge markers.
- Model task and returned product object: model work already happened upstream; the evidence view is primarily a deterministic inspection surface over stored objects.
- Deterministic logic and authority boundary: evidence ordering, citation rendering, and record linkage remain code-governed.
- Persisted state: source records, evidence units, citations, and finding explanations remain visible as durable review artifacts.
- Human review point: the reviewer decides whether the cited evidence really supports the finding or whether the record should be challenged.

## AI dissent review

ATO Bot treats disagreement as a first-class object. When a secondary model or review path dissents from the draft determination, that dissent is preserved and surfaced separately from compliance status. This is important because a control can appear compliant or partial while still carrying a disagreement that deserves human inspection before the result is accepted operationally.

- Runtime purpose: preserve and expose disagreement with the primary finding instead of collapsing the assessment into one silent answer.
- Entry points: AI dissent badges, advanced review surfaces, control detail panels, and challenge workflows.
- Operational inputs: control finding, objective reasoning, evidence packet, challenge note, and dissent prompt context.
- Model task and returned product object: a dissenting rationale or challenge narrative that argues for why the draft outcome may be weak, overstated, or incomplete.
- Deterministic logic and authority boundary: the dissent remains attached to the finding as separate metadata; it does not automatically rewrite the final status.
- Persisted state: dissent records, linked challenge context, and reviewer actions remain durable parts of the assessment record.

- Human review point: the assessor decides whether the dissent changes the control disposition, triggers remediation, or is dismissed with explanation.

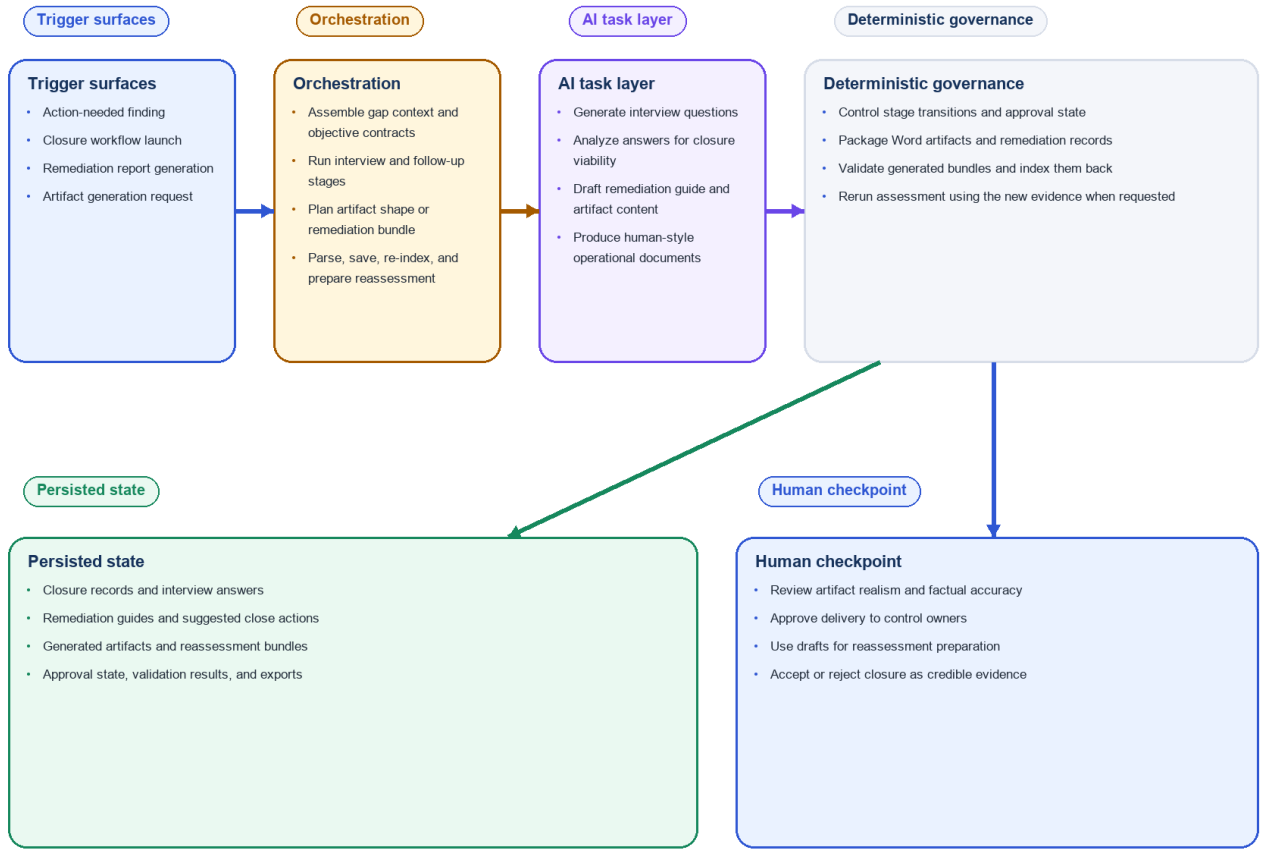
## Manual override, challenge, and reviewer notes

The platform also includes explicit human-authority actions. Reviewers can add notes, open challenge flows, and apply manual overrides where policy or operational judgment requires a different result than the automated path produced. This is a deliberate governance choice: ATO Bot records human intervention rather than pretending the automated result is always final.

- Runtime purpose: let reviewers question, refine, or replace the automated determination under explicit human authority.
- Entry points: selected-control actions, assessor notes, manual status override controls, challenge flows, and review workspaces.
- Operational inputs: current finding, reviewer rationale, evidence context, policy posture, and prior assessment history where relevant.
- Model task and returned product object: optional note drafting or challenge support may assist, but the authoritative action is human, not model-driven.
- Deterministic logic and authority boundary: override state, note persistence, audit timestamps, and attribution are handled by the application.
- Persisted state: reviewer notes, override rationale, and action history become part of the control and assessment audit trail.
- Human review point: this is the human checkpoint; the operator explicitly takes responsibility for the decision.

## Figure 5. Remediation and Artifact Generation Architecture

Post-assessment workflows turn discovered gaps into closure guidance, remediation plans, draft artifacts, and reassessment-ready bundles for human owners to review.




Interpretation: remediation AI does not self-close controls. It packages actionable guidance and draft evidence so humans can move from assessed weakness to reassessment-ready artifacts inside the same governed system.

Figure 5. Remediation and artifact-generation architecture.

## Closure interview flow

Closure starts with guided information gathering. The closure interview flow uses the current finding, discovered gaps, and available evidence to generate targeted questions for the control owner or assessor. Its job is to convert a vague remediation need into explicit facts about what is missing, what the current implementation actually does, and what documentary record would close the gap credibly.

- Runtime purpose: generate control-specific interview questions that clarify the missing facts and records behind a weak finding.
- Entry points: closure actions from the control review surface and remediation preparation workflows.
- Operational inputs: control finding, gap statements, cited evidence, control metadata, and project context.
- Model task and returned product object: structured closure interview questions and prompts tailored to the identified deficiency.
- Deterministic logic and authority boundary: workflow stage, target control, response capture, and artifact linkage remain product-controlled.

- Persisted state: interview prompts and captured responses can become reusable closure records tied to the control.
- Human review point: owners or assessors answer the questions and decide whether the proposed information request is operationally appropriate.

## Closure analysis flow

After interview inputs exist, the closure analysis flow turns them into a control-specific view of what a passing close state must show. This is one of the most important post-assessment AI behaviors in the platform because it translates raw gaps into close-state expectations a control owner can act on. The flow is designed to stay tied to the discovered gap record rather than drifting into generic advisory language.

- Runtime purpose: convert control gaps and owner responses into a passing-evidence target for reassessment.
- Entry points: closure workflow progression, selected-control remediation actions, and review follow-up surfaces.
- Operational inputs: findings, gap statements, interview answers, existing citations, and control objective context.
- Model task and returned product object: closure recommendations, passing-evidence expectations, and targeted narrative explaining the close state.
- Deterministic logic and authority boundary: stage transitions, storage, and linkage to the original finding remain service-layer responsibilities.
- Persisted state: closure analyses become durable control-level guidance objects that can later feed remediation reporting.
- Human review point: reviewers and owners decide whether the proposed close criteria are realistic, sufficient, and aligned to how the control actually operates.

## Remediation guide generation

The remediation guide flow turns findings and objective gaps into practical action guidance. It is richer than a simple gap summary because it can recommend close actions, likely owners, effort expectations, success criteria, collection guidance, and suggested content language tied to the actual discovered deficiency. This is where ATO Bot starts acting like an operational closure system rather than a score-reporting tool.

- Runtime purpose: generate a control-by-control remediation plan that explains how to close the discovered gap.
- Entry points: remediation tabs, selected-control guidance views, report generation, and final remediation report workflows.
- Operational inputs: control findings, objective gaps, closure analysis, evidence expectations, and project-specific system context.
- Model task and returned product object: action plans, owner guidance, success criteria, collection guidance, and suggested close language.
- Deterministic logic and authority boundary: guide assembly is driven from persisted findings and control context rather than open-ended conversation alone.

- Persisted state: remediation guide content can be stored in remediation report records and surfaced back in the app.
- Human review point: control owners and assessors decide what is actually assigned, edited, approved, and executed.

## Remediation artifact generation

Artifact generation goes a step beyond guidance by creating reassessment-ready draft documents. The service reads findings and objective gaps, plans a document type per control or bundle, drafts structured content, and shapes it toward stronger evidence instead of purely assessor-facing closure text. The generated result is a draft operational artifact intended for owner review, not a self-proving substitute for actual implementation.

- Runtime purpose: generate draft evidence artifacts that directly address the gaps identified for one control or a bundle of controls.
- Entry points: remediation artifact actions, closure workflows, package-generation runs, and final report follow-on work.
- Operational inputs: action-needed findings, gap structures, closure recommendations, system context, and generation planning metadata.
- Model task and returned product object: structured document content and draft artifacts tailored to the control or bundle being closed.
- Deterministic logic and authority boundary: packaging, storage, file naming, and later validation remain product-controlled.
- Persisted state: generated artifacts become durable output objects available for download, review, and later reassessment ingestion.
- Human review point: control owners must review, edit, approve, and operationalize the artifact before it is treated as credible reassessment evidence.

## Human-style artifact generation

The human-style artifact generator exists because a technically correct draft can still look synthetic or excessively assessor-facing. This flow plans the artifact shape first, then uses project and system context to produce drafts that read more like operational records a real team would own. The product value is not that the model writes prettier prose; it is that the resulting artifact is more likely to survive owner review and later assessor scrutiny.

- Runtime purpose: generate remediation artifacts that read like plausible human-authored operational documents rather than generic AI closure text.
- Entry points: realism-oriented remediation generation, test package preparation, and owner-facing evidence drafting.
- Operational inputs: control context, project evidence, system narrative, and artifact-shape planning prompts.
- Model task and returned product object: a draft artifact designed to look and read like a human-authored operational record.
- Deterministic logic and authority boundary: runtime routing, file handling, persistence, and later proof validation remain code-governed.

- Persisted state: the artifact becomes part of the generated evidence domain and can be packaged with supporting provenance.
- Human review point: owners still need to validate tone, facts, responsibilities, and operational suitability before use.

### Reporting and export surfaces

The same assessment substrate also drives remediation reporting and final outputs. Findings views, remediation reports, SSP-oriented outputs, OSCAL-oriented packaging, and artifact downloads all consume the same evidence and finding state. This shared data model matters because it reduces drift between what the system assessed, what it told the reviewer to fix, and what it ultimately exported as the record of the engagement.

- Final remediation reporting: aggregate action-needed controls into a single operational output with gaps, close actions, evidence expectations, and suggested artifacts.
- Assessment findings and rollup views in the main app.
- Remediation reports for action planning and ownership.
- SSP Workbench and implementation-statement-oriented outputs.
- OSCAL export runs and report-generation endpoints where applicable.
- Artifact download surfaces for generated evidence packages and supporting material.

**Why this matters**

ATO Bot does not stop at saying a control is weak. It gives the reviewer a governed path from findings review to evidence inspection, dissent handling, remediation guidance, owner-facing artifact drafting, and final reporting. That is what turns the assessment engine into an operational authorization workbench.

## 8. Bounded AI Architecture Across the Platform

By the time a reader reaches this chapter, the operational story should already be clear: ATO Bot performs full assessment, preserves the result for review, and then drives closure and reporting. Section 8 therefore serves a different job. It explains the cross-cutting AI architecture that makes those product behaviors possible without turning the platform into an unbounded agent.

The common pattern across the platform is simple but important. Code assembles a narrow context for one runtime purpose, selects a purpose-specific prompt and model route, parses the response into a structured application contract, applies deterministic validation or policy, stores the result as durable state, and then exposes it through a human-facing workflow. That pattern is what makes the AI layer reviewable.

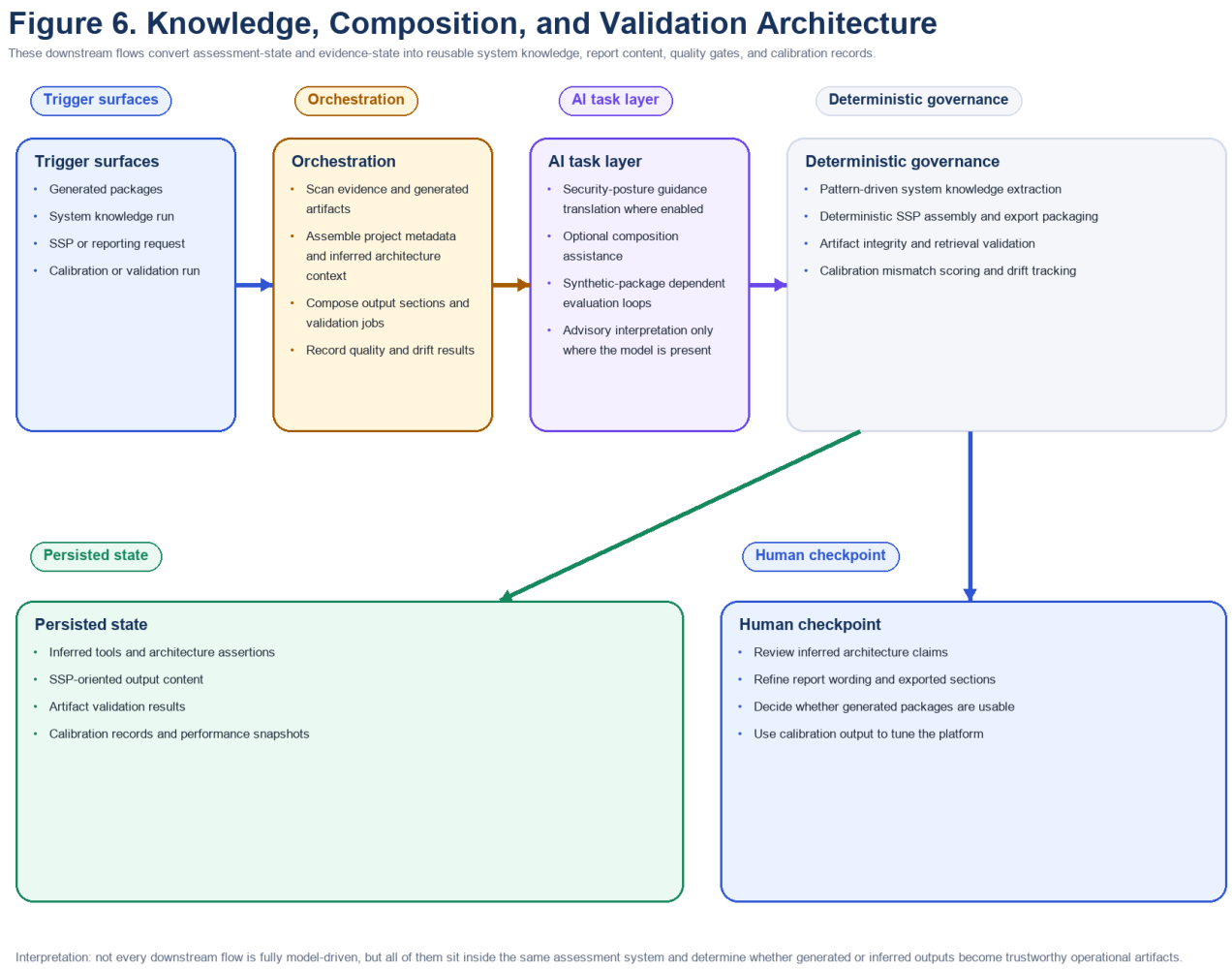



Figure 6. Knowledge, composition, and validation architecture.

## Architecture-wide guardrail pattern

- What AI is allowed to do: classify, reason, summarize, draft, propose, translate, and package within a named runtime purpose.
- What code still decides: scope assembly, schema validation, retry policy, threshold handling, final adjudication, storage, export, and user action boundaries.
- What gets stored: findings, citations, evidence metadata, chats, dissent records, closure guidance, generated artifacts, reports, validation runs, and calibration results.
- Where the human intervenes: review workspaces, dissent inspection, override actions, closure approval, artifact editing, and final report use.
- Why this supports assessment rather than existing for its own sake: every model call is tied to preparing, evaluating, challenging, closing, or packaging an assessment result.

## Evidence preparation band

The evidence-preparation band includes ingestion screening, evidence classification, full-document control tagging, and procedure auto-categorization. Those flows are detailed in Section 5 because they are the front end of the assessment system. Architecturally, their common role is to transform raw files

into typed, reviewable, retrievable evidence objects while preserving provenance and staying clear of final control authority.

- AI allowed to do: identify relevance, classify evidence type, infer likely control support, and sort procedures into operational categories.
- Code still decides: parsing, batching, thresholding, duplicate handling, storage, embeddings, evidence-unit creation, and library placement.
- What gets stored: source records, screening metadata, evidence types, full-document tags, category labels, and retrieval-ready evidence units.
- Human checkpoint: admins and assessors can inspect promoted evidence, correct categories, and later challenge whether the evidence truly supports a control.

## Assessment and adjudication band

The assessment band is the most important AI system in the product. It includes assessment objective evaluation, assessment challenge review, assessment narrative generation, code-governed adjudication handoff, and retry reassessment. Section 6 treated those as product workflows. Architecturally, the key point is that the model is allowed to reason over objectives and evidence, but it is not allowed to assign final policy status without the deterministic adjudication layer.

- AI allowed to do: evaluate objective support, identify contradictions and gaps, draft findings, and rerun the same bounded reasoning contract when retries occur.
- Code still decides: criteria assembly, evidence packet construction, schema validation, policy rollup, override precedence, retry eligibility, and final determination state.
- What gets stored: objective reviews, finding narratives, citations, dissent markers, reviewer-attention signals, retry history, and rollup records.
- Human checkpoint: reviewers inspect findings, review dissents, challenge evidence quality, and apply overrides where policy or judgment requires.

## Review and collaboration band

The review band includes assistant chat purposes, vision attachment interpretation, AI note generation, and dissent collaboration chat. These flows do not decide control posture on their own. Their job is to help a reviewer understand the record, interrogate the result, and move through the assessment with more speed and clarity while keeping authority visible.

- AI allowed to do: answer route-specific questions, interpret attached screenshots, draft short operational notes, and argue through a dissenting position.
- Code still decides: conversation scope, attachment linkage, permissions, note targets, persistence, and whether any generated content becomes part of the record.
- What gets stored: chat transcripts, attachment references, drafted note content where accepted, dissent conversations, and related control links.
- Human checkpoint: reviewers choose what to ask, what to trust, what to save, and whether the discussion changes the official control position.

## Remediation and artifact-generation band

The remediation band includes closure interview flow, closure analysis flow, remediation guide generation, remediation artifact generation, and human-style artifact generation. These capabilities

exist because a post-assessment system must help a team understand not only what failed, but what concrete evidence would close the gap and how to package that evidence into owner-reviewable form.

- AI allowed to do: ask targeted closure questions, translate gaps into passing-evidence expectations, draft remediation guidance, and generate owner-facing evidence artifacts.
- Code still decides: workflow stage, target control, artifact packaging, storage, provenance, validation, and report composition boundaries.
- What gets stored: closure records, remediation guidance objects, generated artifacts, report content, and packaging metadata.
- Human checkpoint: control owners and assessors still validate facts, edit drafts, approve artifacts, and decide whether the close state is credible.

## Knowledge, composition, and validation band

The final band is mixed because some of these capabilities are strongly model-driven while others are deterministic-adjacent but essential to the same system story. The band includes synthetic dataset generation, optional security-posture guidance generation, system knowledge extraction, SSP composition, artifact validation, and the calibration harness. These flows are important because they help the platform test itself, explain itself, and turn assessment outcomes into longer-lived knowledge and reporting products.

## Synthetic dataset generation

The synthetic dataset generator uses the same artifact posture in a test-oriented direction. It extracts project context, builds a system persona, plans target artifact bundles, and generates synthetic but structured evidence packages that can exercise the assessment engine and realism checks.

- Runtime purpose: generate structured evidence bundles that can test assessment behavior without depending only on live customer artifacts.
- Entry points: harness preparation, realism package creation, and internal testing workflows.
- Operational inputs: selected project context, existing evidence profile, and target control or package goals.
- Model task and returned product object: planned bundles and generated artifacts that resemble plausible assessment inputs.
- Deterministic governance: run management, packaging, and downstream harness usage are controlled by worker logic.
- Persisted state: synthetic packages become reusable test assets for assessment evaluation and demos.
- Human checkpoint: reviewers decide whether the synthetic result is realistic enough for calibration or presentation use.

## Optional security-posture guidance generation

Most of this optional security-posture subsystem is deterministic, but when enabled the platform can ask a model to translate structured security findings into clearer operator-facing guidance. This is a bounded translation role, not a supported assessment function or a continuous-authorization capability.

- Runtime purpose: translate structured security-posture or verification observations into operator-facing collection and response guidance.
- Entry points: optional security-posture guidance workflows, verification views, and integration-oriented operator surfaces.
- Operational inputs: structured security finding contracts, severity, observed facts, and control context.
- Model task and returned product object: clearer guidance language and suggested operator-facing framing.
- Deterministic governance: security finding assembly, severity handling, and optional security-posture storage remain code-driven.
- Persisted state: guidance can be attached to optional security-posture workflows or related report surfaces.
- Human checkpoint: operators decide whether the guidance is appropriate for action in their environment.

## System knowledge extraction

System knowledge extraction is strategically adjacent to the agentic stack even though the current implementation is largely deterministic. It reads evidence text to infer likely tools, architecture assertions, and inheritance mappings that can later support architecture views and downstream composition surfaces. The current product truth should be stated plainly: this flow matters, but the AI role is currently limited compared with assessment and remediation.

- Runtime purpose: derive reusable architecture and tooling knowledge from the assessment evidence base.
- Entry points: system knowledge runs, architecture views, and composition-oriented workflows.
- Operational inputs: evidence text, project metadata, parsed records, and existing inferred tool inventory.
- Model task and returned product object: limited or absent today; the current implementation is primarily deterministic.
- Deterministic governance: evidence scanning, assertion persistence, and tool inventory creation are handled by code.
- Persisted state: system knowledge runs, assertions, and inferred tool records become reusable platform objects.
- Human checkpoint: operators review inferred architecture claims before accepting them as authoritative.

## SSP composition

SSP composition also sits closer to deterministic composition today, but it consumes the same evidence, system knowledge, and project metadata foundation. It turns that substrate into reportable SSP sections such as system overview, architecture, tooling, roles, and implementation-oriented narrative blocks. That makes it part of the same bounded AI story even where the current model role is hybrid or limited.

- Runtime purpose: compose structured SSP-oriented output from the same evidence and assessment substrate used elsewhere in the platform.

- Entry points: SSP workbench, reporting flows, and export-generation surfaces.
- Operational inputs: project metadata, findings, system knowledge, implementation statements, and supporting evidence.
- Model task and returned product object: limited or hybrid today; current composition is still largely deterministic.
- Deterministic governance: section assembly, output structure, and packaging are governed by reporting logic.
- Persisted state: SSP-oriented content becomes part of the broader output and export domain.
- Human checkpoint: reviewers refine wording and verify that the composed system description matches actual ownership and evidence.

## Artifact validation

Artifact validation is the quality-control counterpart to artifact generation. It checks whether generated packages are intact, ingestible, mappable to controls, and viable for retrieval. Even though the validator is deterministic, it is central to trust because it prevents generated evidence from bypassing the same operational gates as any other document.

- Runtime purpose: verify that generated or assembled evidence packages are structurally usable for later assessment work.
- Entry points: artifact package review, remediation package validation, and harness preparation workflows.
- Operational inputs: generated files, manifests, ingestion results, control mappings, and retrieval viability checks.
- Model task and returned product object: none in the current implementation.
- Deterministic governance: integrity checks, ingestion completion, control mapping, and retrieval viability are all scored by code.
- Persisted state: validation runs and results remain available for package review and quality tracking.
- Human checkpoint: operators can see whether a generated package is merely present or actually usable for reassessment work.

## Calibration harness

The calibration harness evaluates AI-driven assessment behavior against known synthetic-package expectations. It compares expected outcomes with actual results, records mismatch patterns, and surfaces drift types. This is less a user-facing assistant than an engineering loop for maintaining assessment reliability over time.

- Runtime purpose: measure whether AI-assisted assessment behavior remains aligned with expected outcomes over repeated test runs.
- Entry points: harness executions, family-package runs, realism validation, and regression testing workflows.
- Operational inputs: synthetic or curated test bundles, expected control outcomes, actual findings, and mismatch metadata.
- Model task and returned product object: indirect; the harness evaluates AI-driven outputs rather than acting as a model workflow itself.

- Deterministic governance: package comparison, mismatch scoring, drift tracking, and performance snapshots are code-governed.
- Persisted state: calibration outcomes become long-lived quality records for future tuning and regression analysis.
- Human checkpoint: engineers and reviewers inspect drift over time instead of assuming the model layer stays stable automatically.

**Why this matters**

The strength of ATO Bot is not that it has many model calls. It is that every important model behavior has a narrow runtime purpose, a code-defined authority boundary, durable state, and an obvious place for human review. That architecture is what lets the platform behave like a governed assessment system instead of a generic AI document tool.

# 9. Operational Observability and Optional Security-Posture Surfaces

ATO Bot records operational-observability data around the supported assessment workflow, including service health, job state, audit activity, configuration changes, and processing failures. The codebase also retains a separately gated integration and security-posture subsystem with objects for integration accounts and runs, posture snapshots, drift records, security collectors and assets, and verification results.

The supported observability layer helps operators determine whether the assessment workflow is available and trustworthy. The optional security-posture subsystem is not part of the supported evidence-to-assessment claim and must not be presented as continuous authorization monitoring.

- Integrations page for connecting external systems and future data feeds.
- Operational-observability records for service, job, audit, configuration, and processing status.
- Optional security-posture guidance generation for translating structured findings into collection targets.
- Gated verification, drift, and snapshot objects retained for future security-posture work.

The most credible way to position the supported layer is as operational observability around the assessment record. It tells operators whether processing is current and whether audit and workflow state are available. The separate security-posture subsystem remains a disabled expansion lane rather than a claim of continuous authorization.

# 10. Trust, Governance, and Differentiators

ATO Bot's primary differentiator is not that it uses larger models. It is that the surrounding architecture gives the models bounded jobs inside a persisted, reviewable, policy-aware system. Trust comes from design choices that make evidence and reasoning inspectable rather than from claims that the AI is universally correct.

## Governance strengths

- Purpose-scoped runtime routing rather than undifferentiated prompt calls.

- Evidence provenance preserved from parsed source records through derived evidence units and final citations.
- Assessment policy, buckets, overrides, and code-based control rollup instead of free-form status assignment.
- Human review, challenge, dissent, override, and remediation workflows embedded in the same app.
- Persistent reporting, audit, export, and admin surfaces that make the system operational rather than purely generative.

### Key limitations to state plainly

- The platform surface is broad, so presentation and information architecture must stay disciplined to avoid operator overload.
- Some ingestion and retrieval compatibility layers still reflect an older chunk-first model and will continue to need cleanup over time.
- Optional security-posture monitoring and some system-knowledge flows are meaningful but not yet as mature as the core assessment experience.
- Because the platform spans many jobs, reliability depends on background orchestration, persistence hygiene, and runtime configuration discipline as much as on model quality.

Those limitations do not weaken the central thesis; they clarify where current strength already exists and where the expansion path should be framed carefully.

## 11. Conclusion

ATO Bot is best understood as a governed evidence operations system for NIST 800-53 authorization work. It ingests and normalizes source artifacts, converts them into retrievable evidence units, applies purpose-scoped model tasks, adjudicates findings through policy-aware code, and keeps the human assessor in authority over review, remediation, and reporting.

That architecture gives the product a stronger technical posture than systems that jump directly from upload to narrative. The platform is already broad enough to support document libraries, assessment execution, remediation, SSP and OSCAL outputs, system knowledge, optional integrations, and operational observability. Its core differentiator is therefore not generic AI capability but the combination of provenance, bounded orchestration, deterministic governance, and operationally useful review surfaces.

# Appendix A. Full App Surface Inventory

The current route structure below focuses on the ATO workflow and the supporting admin and evidence-management surfaces described in this paper, confirming that ATO Bot is a multi-surface operations application rather than a single-page assessment tool.

| Route | Component | Purpose |
|---|---|---|
| /login | Login | Application route |
| / | Navigate | Application route |
| /dashboard | Navigate | Application route |
| /projects | ProjectList | Application route |
| /projects/:id | ProjectDetail | Application route |
| /projects/:id/integrations | OptionalSecurityPostureRoute | Application route |
| /projects/:id/architecture-tools | SystemKnowledgePage | Application route |
| /projects/:id/calibration | CalibrationPage | Application route |
| /assessment-policy | AssessmentPolicyPage | Application route |
| /projects/:id/ssp-workbench | SspWorkbenchPage | Application route |
| /projects/:projectId/assessments/:assessmentId | AssessmentView | Application route |
| /projects/:id/audit-log | ProjectAuditLog | Application route |
| /projects/:id/test-dataset | TestDatasetPage | Application route |
| /users | UserManagement | Application route |
| /security/audit-log | AuditLog | Application route |
| /admin/dashboard | SecurityDashboard | Application route |
| /admin/prompts | PromptManager | Application route |
| /admin/ai-runtime | IngestionConfig | Application route |
| /admin/ingestion-config | Navigate | Application route |
| /common-controls | CommonControls | Application route |
| /enterprise-policies | EnterprisePolicies | Application route |
| /enterprise-procedures | EnterpriseProcedures | Application route |
| /control-catalog | ControlCatalogPage | Application route |
| * | Navigate | Application route |

# Appendix B. Content Libraries and Evidence Domains

- Project library: system-specific artifacts, uploads, and generated outputs scoped to an assessed system boundary.
- Common-control library: inherited or provider-owned evidence that can support multiple projects.
- Enterprise policy library: organization-level governance documents reused across assessments.
- Enterprise procedure library: operational procedures, review records, and execution guidance reused across controls or projects.
- Generated artifact domain: remediation artifacts, test packages, human-style evidence drafts, SSP outputs, and report exports.

These domains matter because ATO Bot does not assume one folder of documents equals one complete assessment corpus. It treats evidence as layered and reusable across different responsibility boundaries.

# Appendix C. Service and API Map

Backend API modules show the breadth of the application contract exposed to the frontend and operator workflows.

| API module | Category |
|---|---|
| activity_log | API module |
| admin | API module |
| admin_prompts | API module |
| ai_assist | API module |
| artifacts | API module |
| assessment_governance | API module |
| assessment_policy | API module |
| assessments | API module |
| assistant | API module |
| auth | API module |
| calibration | API module |
| closure | API module |
| common_controls | API module |
| control_catalog | API module |
| documents | API module |
| enterprise_policies | API module |
| enterprise_procedures | API module |
| ingestion_config | API module |
| integrations | API module |
| llm | API module |
| meta | API module |
| overrides | API module |
| projects | API module |
| remediation | API module |
| reports | API module |
| ssp | API module |
| system_knowledge | API module |
| system_profile | API module |
| test_dataset | API module |
| users | API module |

Service modules and service subpackages reveal the main internal orchestration and domain layers.

| Service | Type |
|---|---|
| activity_log | Service module |
| artifact_generator/ | Service package |
| artifact_validation | Service module |
| assessment_engine | Service module |
| assessment_governance | Service module |
| assessment_pipeline | Service module |
| assessment_policy | Service module |
| assessment_rollup | Service module |
| assistant_service | Service module |

| | |
|---|---|
| ato_bot_security | Service module |
| calibration_harness | Service module |
| closure_guidance | Service module |
| closure_service | Service module |
| control_tagger | Service module |
| controls/ | Service package |
| evidence_profiles | Service module |
| evidence_quality | Service module |
| evidence_view | Service module |
| human_artifact_generator | Service module |
| implementation_statements | Service module |
| ingestion/ | Service package |
| integrations | Service module |
| job_worker | Service module |
| llm/ | Service package |
| multistage_engine | Service module |
| package_generation | Service module |
| package_usefulness | Service module |
| parsers/ | Service package |
| procedure_categorizer | Service module |
| prompt_manager | Service module |
| rag/ | Service package |
| remediation_service | Service module |
| reports/ | Service package |
| retry_engine | Service module |
| scorecard/ | Service package |
| security_telemetry | Service module |
| ssp_composer | Service module |
| system_knowledge | Service module |
| system_profile | Service module |
| test_dataset_generator | Service module |
| tool_guidance | Service module |

# Appendix D. Data Model Snapshot

| Domain | Representative objects |
|---|---|
| Identity and access | User, RefreshToken, AuditLog |
| Project and document domain | Project, Document, PolicyLibrary, ProcedureLibrary, CommonControlProvider, ProjectCommonProvider |
| Ingestion pipeline v2 | IngestionRun, ParsedDocumentRecord, ParsedLine, ScreeningResult, EvidenceUnit, EvidenceClassification, EvidenceEmbedding |
| Assessment domain | Assessment, ControlFinding, AssessmentCriteriaPackage, AssessmentEvidenceTriage, ObjectiveEvidenceReview, ObjectiveDetermination, ControlDetermination, |

| | |
|---|---|
| | AssessmentChallenge, AssessmentRollup, ControlOverride, ControlActivityLog |
| Governance and policy | AssessmentPolicy, AssessmentPolicyBucket |
| Remediation and reporting | RemediationReport, POAM, OscalExportRun |
| Assistant and collaboration | AssistantConversation, AssistantContextAttachment, AssistantMessage |
| System knowledge and validation | SystemKnowledgeRun, SystemKnowledgeAssertion, ToolInventory, ArtifactValidationRun, ArtifactValidationResult, PackageViabilityRun |
| Optional security-posture data | IntegrationAccount, IntegrationRun, TelemetrySnapshot, ControlTelemetryPosture, DriftRecord, SecurityCollector, SecurityAsset, SecurityScan, SecurityFinding, SecurityRecommendation, VerificationCheck, VerificationResult |

# Appendix E. Outputs, Reports, and Exports

- Assessment findings workspaces and rollup summaries.
- Remediation reports and closure-oriented guidance.
- Generated remediation artifacts and human-style evidence artifacts.
- SSP Workbench content and implementation statement outputs.
- OSCAL export runs and report download surfaces.
- Synthetic dataset packages, validation artifacts, and package viability outputs where used.

A notable design advantage is that these outputs draw from the same underlying evidence and finding state as the assessment engine, reducing the likelihood that exported reports drift from review-time truth.

# Appendix F. Current-State Capabilities vs Expansion Path

| Area | Interpretation |
|---|---|
| Most mature today | Evidence ingestion, normalization, retrieval, assessment execution, adjudication, review, remediation guidance, reporting surfaces |
| Strong supporting surfaces | Prompt/runtime admin, document libraries, common-control reuse, assistant workflows, synthetic dataset generation |
| Strategic expansion path | Optional security-posture correlation, deeper integration automation, and broader verification loops |
| Presentation caution | The product surface is broad enough that technical communications should lead with evidence-to-assessment first and move future-state surfaces into supporting sections or appendices |