When Agents Need the Real World: Verifiable Human Execution for Reality-Bound Agent Tasks

A System Model, Prototype, and Evaluation Plan for AI2Human

AI2Human Research

June 27, 2026

Abstract

Autonomous agents can browse, call APIs, write code, and coordinate digital workflows, but they still fail when a workflow requires someone to enter a store, inspect a shelf, verify a delivery, collect a signature, or produce fresh evidence from a physical location. Today these reality-bound steps are handled through informal coordination: a user asks someone in chat, receives an unstructured photo or message, manually decides whether it is sufficient, and pays through a separate channel. This breaks the agent workflow exactly where reliability matters: the task is underspecified, proof is ambiguous, verification is inconsistent, payment is detached from completion, and the history is hard to audit.

This paper introduces **Structured Human Execution Infrastructure**, a system layer that lets agents turn a blocked real-world step into a bounded human-executable task, receive structured proof, verify the result, and release payment only after acceptance. AI2Human is our prototype instance of this layer. The execution loop is: task specification, human execution, proof submission, verification against task constraints, and conditional settlement through on-chain USDC or equivalent auditable payment rails. We formalize the key objects in this loop, including `RealityBoundTask`, `ProofRequirement`, `ProofBundle`, `VerificationDecision`, `SettlementCondition`, and `GovernancePolicy`.

The contribution is not that humans can perform field tasks for payment; field task platforms already demonstrate that. The contribution is an agent-native interface in which tasks, proof requirements, verification state, settlement state, and audit logs are machine-readable and can be reintegrated into an agent workflow. We further outline **RealityBound-220**, a benchmark and evaluation plan for measuring agent recognition of reality-bound steps, proof requirement quality, verification reliability, settlement correctness, privacy-preserving proof design, governance effectiveness, and operator availability limits.

1 Introduction

Autonomous agents are increasingly able to operate inside digital systems. They can browse websites, call APIs, edit documents, write code, use software tools, and coordinate multi-step workflows across online services. This progress has made it natural to imagine agents as increasingly general-purpose workers. Yet the boundary of agent autonomy is not only a matter of model intelligence or tool access. Many useful workflows cross into the physical, social, institutional, and identity-bound world. At that boundary, agents encounter steps that cannot be completed by a browser, an API, or a language model.

Consider a purchasing agent asked to find the best local option for a product. The agent may compare online prices, parse reviews, draft a recommendation, and prepare a purchase plan. But it cannot walk into a specific store, confirm whether the item is actually on the shelf, photograph the

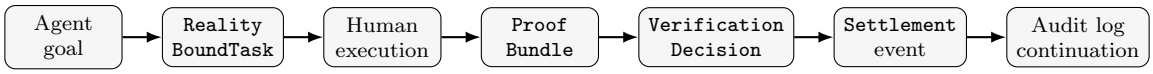

Figure 1: Structured human execution loop for reality-bound agent tasks. The key property is that proof, verification, settlement, and audit state remain explicit rather than disappearing into informal coordination.

shelf with a fresh timestamp, ask a clerk for confirmation, and return evidence that another system can verify before releasing payment. Similar constraints appear in many workflows: confirming event attendance, collecting a physical signature, checking whether a poster is installed, verifying a delivery, inspecting a device, documenting a local condition, or completing an identity-bound action that requires a real human account or presence.

These are not merely temporary failures caused by weak models. Some steps require embodiment, location, human identity, local perception, social interaction, legal authority, or evidence collection in the real world. A more capable model may better recognize such constraints, plan around them, and specify what must be done, but it still cannot itself become physically present, socially authorized, or institutionally recognized in every context. As autonomous agents enter workflows that touch the real world, they need infrastructure for delegating reality-bound steps to humans in a way that is structured, verifiable, auditable, and economically settled.

Current recovery paths are often informal. A user or operator sends a message to someone nearby, asks for a photo or confirmation, manually judges whether the result is trustworthy, and pays through a separate channel. This approach may work for one-off coordination, but it does not scale into agent infrastructure. Informal coordination leaves task scope ambiguous, proof requirements underspecified, verification decisions inconsistent, payment detached from completion, and the event history difficult for agents or auditors to reconstruct. A workflow that disappears into private chats and manual payments cannot be reliably resumed by an autonomous system.

This paper proposes **Structured Human Execution Infrastructure**: a system layer that converts reality-bound workflow steps into human-executable tasks. Each task carries explicit constraints, proof requirements, verification rules, settlement conditions, and governance policies. A human operator completes the real-world step and submits a structured proof bundle. A verifier or verification engine evaluates that proof against the original constraints. Settlement is released only after acceptance, through on-chain USDC or an equivalent auditable payment rail. The resulting event history records what was requested, who claimed the task, what proof was submitted, how it was verified, and why payment was released, refunded, disputed, or withheld.

The core loop is:

`task -> human execution -> proof -> verify -> settle`

This loop is simple, but each term has system-level consequences. A `task` must be specific enough for a human to execute and for an agent to track. `Human execution` must be bounded by safety, consent, compensation, and policy constraints. `Proof` must be structured enough to support automated, model-assisted, or human verification. `Verification` must distinguish deterministic checks from contextual judgments. `Settlement` must be economically viable for low-value tasks while remaining auditable for higher-value or disputed tasks.

This paper is not claiming that field work, local task execution, or photo-based proof is new. Field task platforms and gig execution networks have long coordinated people to visit physical locations, inspect retail conditions, take photos, and receive payment. The difference is the system boundary. Existing field task platforms are generally designed for human operations teams and business requesters. Structured Human Execution Infrastructure is designed for autonomous agents and agentic workflows: tasks are agent-addressable, proof requirements are machine-readable, ver-

ification is explicitly modeled, settlement is tied to verification state, and the entire loop can be audited and reintegrated into an agent's workflow.

The paper should be read as a systems paper, not as a company white paper or go-to-market document. AI2Human is used as a prototype system and motivating implementation, while RealityBound-220 is used as an evaluation artifact. Where the current prototype implements a component, we identify it as implemented. Where the paper describes a general abstraction or later-stage capability, we separate that from the M1 system boundary.

The paper makes five contributions.

First, we define **reality-bound task delegation**, a class of agent workflow steps that require physical presence, local perception, human identity, institutional interaction, object manipulation, or real-world evidence collection. These steps are distinct from ordinary reasoning failures or missing API calls because they require capabilities outside the agent's digital tool boundary.

Second, we introduce **Structured Human Execution Infrastructure**, a system model that converts reality-bound steps into human-executable tasks with explicit constraints, operator requirements, proof schemas, verification rules, settlement conditions, and governance policies, and we instantiate the model in the AI2Human prototype.

Third, we formalize a **proof-centered verification model** that relates task constraints, proof requirements, submitted proof bundles, automated checks, human verifier decisions, and final task acceptance.

Fourth, we describe an **on-chain settlement layer** that connects verified completion to escrowed USDC settlement, release/refund conditions, dispute states, and auditable payment records.

Fifth, we propose **RealityBound-220**, a benchmark and evaluation framework for comparing structured execution infrastructure with informal human coordination. The evaluation measures agent recognition of reality-bound steps, task specification quality, proof requirement over- and under-provisioning, verification reliability, settlement correctness, privacy-preserving proof design, governance effectiveness, and operator availability limits.

2 Background and Related Work

Structured Human Execution Infrastructure sits between human computation, AI delegation, agent benchmarks, field task marketplaces, provenance, and programmable settlement. Prior work supplies important pieces, but no existing line of work covers the full loop studied here: agent-addressable task specification, human real-world execution, structured proof, verification-linked settlement, auditability, and governance.

2.1 Human-in-the-loop AI and human computation

Human-in-the-loop AI studies how people supervise, label, correct, evaluate, or approve machine outputs. Human computation and crowdsourcing systems, from Mechanical Turk to CrowdForge, TurKit, Flash Teams, and CrowdDB, treat human work as a computational resource and analyze decomposition, redundancy, aggregation, quality control, incentives, fairness, and privacy [3, 4, 6, 1].

These systems provide foundations for task quality and worker reliability, but the unit of work is different. In our setting the output is not mainly a label, ranking, or answer. It is a proof bundle that demonstrates a constrained real-world action occurred: a shelf photo, signed receipt, event check-in, delivery confirmation, location-bound observation, or identity-bound action. The request may originate from an autonomous agent, and the result must be verified, settled, and returned to the agent workflow.

2.2 Humans as callable resources in agent workflows

Recent work such as **Human Tool: An MCP-Style Framework for Human-Agent Collaboration** models humans as callable interfaces that agents can invoke for capabilities, information, or authority. Our focus is narrower: humans are executors of reality-bound tasks whose outputs must be structured as evidence. The central question is not only when an agent should ask a person, but how the task should be specified, what proof should be required, how proof should be verified, and when payment should be released.

2.3 Intelligent AI delegation and agent coordination

AI delegation research studies handoff, authority, responsibility, monitoring, verifiability, reversibility, and accountability among humans, AI systems, and agents. Agent protocols and tool-use systems similarly define how agents call external capabilities. We treat structured human execution as one concrete branch of this space: reality-bound delegation, where the missing capability is physical presence, human identity, institutional interaction, object manipulation, or evidence collection. This branch requires proof requirements, verification rules, settlement conditions, and governance policies that general delegation frameworks usually leave unspecified.

2.4 Physical and embodied human-AI collaboration

Assistive agents, augmented reality guidance, embodied instruction-following, and human-robot interaction study how AI systems help people perform physical tasks. Our setting reverses the direction: an autonomous agent creates a bounded task because its workflow needs a reality-bound step. The human operator produces proof, the system verifies it, and settlement follows verification. Proof, auditability, and economic state are therefore central rather than peripheral.

2.5 Agent audit, provenance, and verifiable execution

Agent audit and provenance research records tool calls, decisions, action histories, permission boundaries, tamper-evident logs, and user oversight. Structured human execution adds a complementary history: what happened after an agent assigned a reality-bound step to a person. The audit log must record task creation, claim, proof submission, checks, verifier decisions, settlement events, disputes, and policy outcomes. This extends provenance from digital action histories into agent-human execution histories.

2.6 Agent benchmarks and autonomy measurement

Benchmarks such as WebArena, OSWorld, TheAgentCompany, and SWE-agent evaluate realistic digital work: web navigation, operating-system control, software engineering, document manipulation, and simulated organizations [11, 8, 9, 10]. They show that agents struggle in long, tool-heavy, contextual workflows. Most remain inside digital environments. RealityBound-220 measures a different boundary: whether agents recognize reality-bound steps, specify human-executable tasks, request sufficient proof, and continue after verification and settlement.

2.7 Field task verification marketplaces and gig execution platforms

Commercial platforms such as Gigwalk, Field Agent, EasyShift, Streetbees, and TaskRabbit already coordinate people to visit locations, collect photos, check inventory, verify displays, report prices,

and receive payment. They are crucial comparators because they prove that field execution, photo proof, and payout workflows are commercially real.

Our contribution is not that local human work can be coordinated. It is to make the pattern agent-native. In field task marketplaces, tasks are usually authored and managed by human operations teams, brands, or research managers. In Structured Human Execution Infrastructure, tasks are invoked by agents; task schemas, proof requirements, verification rules, settlement conditions, and audit logs are machine-readable; proof is designed for automated, model-assisted, or human verification; settlement is bound to verification state; and the result returns to the agent as a verified state transition rather than an informal answer.

2.8 Web3 payments, escrow, and agentic settlement

Payment is part of the trust model. Stripe Connect and PayPal Commerce Platform provide mature marketplace payout, compliance, and dispute infrastructure, but payment state is platform-mediated and not usually exposed as a task-level audit trail for agents. Smart-contract escrow and ERC-20 stablecoins such as USDC offer a different point in the design space [7]: funds can be locked, release/refund conditions represented as state transitions, and settlement events emitted for audit. We do not argue that every task should settle directly on-chain; batching and hybrid settlement may be necessary. The key requirement is that economic state remains linked to verification state.

3 Problem Definition: Reality-Bound Agent Steps

This paper studies **reality-bound task steps**: steps in an agent workflow that require physical presence, local perception, object manipulation, human identity, social or institutional interaction, or real-world evidence collection outside the agent's digital tool boundary. They differ from ordinary model, API, or tool failures because the missing capability is contact with a part of the world the agent cannot directly access.

These steps matter because they can appear inside otherwise digital workflows. An agent may complete most of a task online, then become blocked by one physical or institutional requirement. Without a structured handoff, the workflow either fails or moves into informal coordination, leaving the task scope, proof standard, verification decision, and payment state ambiguous.

We classify reality-bound task steps into six categories.

3.1 Physical presence

Some tasks require a person at a location: checking whether a store is open, confirming whether an event booth exists, inspecting a public notice, or verifying pickup availability. Proof may include a timestamped photo, location metadata, short video, or observation note.

3.2 Local perception

Some tasks require observation from a specific vantage point at a specific time: shelf condition, sign text, product labels, display installation, queue length, device state, poster presence, or package condition. A request for "send a photo" is often insufficient; proof must specify what must be visible, what context is needed, and whether freshness, nonce, or operator notes are required.

3.3 Object manipulation

Some steps require handling objects: pickup, delivery, placement, document collection, component installation, or device checks. Proof may require before/after evidence, signed receipts, delivery confirmations, or video. These tasks raise safety and governance concerns around illegal goods, dangerous work, licensing, travel burden, and scope expansion.

3.4 Identity and authority

Some tasks require human identity, account ownership, authorization, or signature: signing for a delivery, confirming attendance, binding a social account, or verifying membership. These tasks are privacy-sensitive and vulnerable to coercion, impersonation, and overcollection, so proof design should use selective disclosure, redaction, verifier-only access, and clear retention rules.

3.5 Social and institutional interaction

Some tasks require interaction with people or institutions: asking a clerk about stock, checking in at a venue, confirming a front-desk policy, or collecting a stamped document. Proof may depend on context that automated checks cannot fully decide, so these tasks often require human or governance review.

3.6 Evidence and accountability

Some tasks are primarily about evidence: documenting damage, verifying signage installation, confirming poster removal, or recording that a condition existed at a specific time. The action may be simple, but the proof must support later inspection. If evidence is insufficient, the agent cannot safely treat the task as complete.

3.7 Why informal coordination is insufficient

Informal coordination mixes five concerns that should be separate: task scope, execution, proof, verification, and payment. A chat message may contain the task but not a specification; a photo may contain evidence but not a proof schema; a manual approval may settle work without explaining why it was accepted; payment may move without being linked to verification state. Structured Human Execution Infrastructure separates these concerns so agents, requesters, verifiers, operators, and auditors can reconstruct what happened.

4 System Model: Structured Human Execution Infrastructure

Structured Human Execution Infrastructure exposes reality-bound human work as an execution interface for agents. The goal is not merely to route a message to a person. The system must preserve workflow semantics: what was requested, what constraints apply, what proof is required, how completion is verified, what economic state changes after acceptance, and how the event history can be audited.

The system contains seven actors. The **Agent / Requester** creates a reality-bound task. The **Task Router** matches it to operators by location, capability, risk, deadline, and reliability. The **Human Operator** performs the action and submits proof. The **Verifier** evaluates proof using rules, model-assisted checks, human judgment, or governance escalation. The **Settlement Contract / Payment Layer** holds and releases reward according to verification. The **Governance**

Layer enforces task policy, disputes, operator protections, and abuse prevention. The **Audit Log** records task, claim, proof, verification, settlement, and governance events.

These actors operate over explicit objects. A `RealityBoundTask` specifies objective, context, deadline, reward, risk, and constraints. `OperatorCapability` describes coverage, task categories, reliability, language, qualification, and safety limits. `ProofRequirement` describes artifacts, metadata, acceptance criteria, privacy rules, and verification rules. `ProofBundle` contains submitted photos, videos, notes, receipts, timestamps, location metadata, account screenshots, or transaction records. `VerificationDecision` records acceptance, rejection, revision, dispute, or governance review. `SettlementCondition` determines release, refund, lock, or batch payout. `GovernancePolicy` defines task categories, prohibited use, privacy requirements, operator rights, and escalation paths.

This object model is intentionally narrow. It represents the parts of human execution an agent workflow must understand to delegate, verify, settle, and continue. In this sense, it is closer to an agent-computer interface than to a generic marketplace.

4.1 Execution loop

The execution loop is:

`agent goal -> reality-bound task -> operator execution -> proof bundle -> verification -> settlement -> audit log -> agent workflow continues`

The loop begins when an agent identifies a reality-bound step and creates a task. A purchasing agent, for example, may request confirmation that Product X is available at Store A before 6pm, with a 10 USDC reward, shelf photo, and operator note. The router selects an operator by geography, reliability, task category, deadline, availability, and policy constraints. The operator submits a proof bundle. The verifier checks artifacts, time window, relevance, metadata, and consistency. If accepted, settlement releases payment or records the task for batch payout, and the audit log returns a verified result to the agent.

4.2 Walkthrough: local inventory verification

The following trace illustrates the system end to end.

1. **Task creation.** The agent determines that online inventory data is insufficient and creates a `RealityBoundTask`: "Confirm whether Product X is available at Store A." The task includes store location, deadline, reward, proof requirements, settlement condition, and risk level.
1. **Operator assignment.** The task router selects an operator with local coverage and sufficient reliability. The task is low-risk because it involves public retail information and does not require private access, identity proof, or high-value transfer.
1. **Human execution.** The operator visits Store A, checks the relevant shelf, optionally asks a clerk, and captures evidence.
1. **Proof submission.** The operator submits a shelf photo, timestamp, optional GPS metadata, and an operator note.
1. **Verification.** The rule engine checks required artifacts, submission window, file hash uniqueness, and metadata consistency. A human verifier confirms that the photo shows the target shelf and that the note matches visible evidence.
1. **Settlement.** If proof is accepted, payment is released or included in a batch payout. If proof is insufficient, the task enters `needs_revision`, `rejected`, or `disputed`.
1. **Audit and continuation.** The audit log records the task, claim, proof hash, verification decision, and settlement event. The agent receives verified stock status and continues the original purchase recommendation workflow.

The walkthrough highlights the systems distinction: a chat request may get a photo back, but it does not capture task constraints, proof requirements, verification criteria, settlement state, and audit history in a form an agent can use.

4.3 Task state and completion

The system distinguishes proof acceptance from economic completion. `Accepted` means the submitted proof satisfies verification requirements and is sufficient for the agent workflow. `Released` means the proof has been accepted and the operator has received full payment. For evaluation, the primary success metric should be `released / successfully completed`, not merely accepted proof.

Other states prevent evaluation from overstating success. `Needs revision` means proof is incomplete but repairable. `Rejected` means the task, action, proof, or policy failed. `Refunded` returns reward to the requester. `Disputed` is unresolved and should be reported separately. `Closed` is terminal after release, refund, cancellation, or dispute resolution. For agent workflows, completion means the agent can safely continue and economic state has settled according to policy.

5 Proof-Based Human Execution

The central interface between human execution and agent workflows is proof. The operator does not simply say a task is complete; they return a structured proof bundle that allows the system to decide whether a reality-bound action occurred under specified constraints. This shifts the unit of work from hiring a person to producing a verifiable outcome: what action occurred, what evidence demonstrates it, what metadata is needed, and what evidence would be excessive or privacy-invasive.

5.1 Proof types

Proof requirements depend on task type. We group proof into six broad categories.

Visual proof includes photos, videos, or screen recordings for inventory, attendance, installation, delivery, and condition documentation. It is intuitive but vulnerable to reuse, synthesis, cropping ambiguity, and staged scenes.

Location proof includes GPS metadata, geofence checks, map screenshots, or device location evidence. It supports presence claims but is not sufficient alone because it can be spoofed and does not prove action.

Temporal proof includes server timestamps, device timestamps, time-window checks, or freshness constraints. It prevents stale evidence but must handle device clocks, upload delays, and time zones.

Transactional proof includes receipts, order numbers, logistics IDs, platform confirmations, or on-chain transactions. It is structured but may expose sensitive information.

Identity-bound proof includes signatures, account-bound screenshots, membership confirmations, or actions requiring a specific account or person. It should use minimal disclosure.

Third-party proof includes clerk confirmation, institutional stamps, public webpage status, or platform records. It can improve reliability but is hard to standardize.

5.2 Proof quality

A proof bundle should be evaluated along several dimensions.

Completeness asks whether all required artifacts are present. **Relevance** asks whether the evidence directly supports the task objective. **Freshness** asks whether the evidence was generated within the task window. **Authenticity** asks whether the evidence appears reused, forged, manipulated, or inconsistent. **Constraint alignment** asks whether the proof satisfies the original task constraints rather than merely showing related evidence. **Privacy compliance** asks whether the proof avoids unnecessary personal, location, or account information. **Reviewer agreement** asks whether independent verifiers reach the same conclusion.

These dimensions should be visible to agents. If a shelf photo, timestamp, and note are enough, a selfie or full identity document is unnecessary. Proof quality affects task generation, operator burden, privacy, cost, and dispute rates.

5.3 Minimal sufficient proof

For each task, the system should define a **minimal sufficient proof set**: the smallest artifact set that allows reliable verification without unnecessary privacy exposure or operator burden. This set can be expert-labeled for benchmark tasks and encoded as production templates.

For example:

```
Task: Confirm item is in stock.
Minimal proof: [photo of shelf + timestamp]
Over-provisioned request:
  [photo of shelf + timestamp + operator selfie + store front photo + GPS + store receipt]
Over-provisioning score:
  unnecessary items / total requested items = 4 / 6 = 66.7%
```

Under-provisioning omits evidence needed for reliable verification; over-provisioning requests unnecessary evidence. A weak requirement makes verification unreliable, while a maximalist requirement makes the system invasive, costly, and less usable. The design target is sufficient proof, not maximal proof.

5.4 Proof specificity and under-specification

The phrase "photo proof" is too weak. A useful `ProofRequirement` must specify artifact type, capture constraints, metadata constraints, freshness window, anti-replay requirement, privacy rules, and verification rule. Under-specification is a first-class failure mode: a task can request the right broad proof type while still producing unverifiable evidence.

A structured proof requirement has the following fields:

```
{
  "artifact_type": "photo",
  "target": "Product X shelf area",
  "capture_constraints": {
    "required_context": ["shelf label", "adjacent products", "task nonce"],
    "angle": "front-facing shelf view",
    "minimum_lighting": "text on shelf label must be legible",
    "occlusion_rule": "target area must not be blocked by hand or bag"
  },
  "metadata_constraints": {
    "server_received_before_deadline": true,
    "gps_tolerance_meters": 50,
    "freshness_minutes": 30
```

```
  },
  "anti_replay": {
    "task_nonce_visible": true,
    "duplicate_hash_rejected": true
  },
  "privacy_constraints": {
    "redact": ["faces", "payment cards", "private labels"],
    "forbidden_content": ["bystander close-ups"]
  },
  "verification_rule": "accept if shelf area, label, nonce, and timestamp constraints are satisfied"
}
```

The schema also handles negative results. If Product X is unavailable, valid evidence may be a photo of the empty shelf area, shelf label, adjacent category context, and an operator note. The outcome is not failed execution; it is successful verification of absence.

5.5 Proof as an agent-facing contract

Proof requirements act as a contract between agent, operator, verifier, and settlement layer. The agent knows what evidence to expect, the operator knows what to submit, the verifier knows what criteria to apply, and settlement knows what decision releases payment. Unlike informal coordination, the proof requirement exists before work begins, reducing ambiguity and making disputes easier to resolve.

6 Verification: Automated Checks and Human Judgment

Verification determines whether a proof bundle satisfies the task constraints. It is tempting to imagine verification as fully automated, but this is unrealistic for many reality-bound tasks. Some checks are deterministic. Some rely on metadata. Some can be assisted by models. Some require human judgment. Some require governance escalation because the task is high-risk, privacy-sensitive, contested, or potentially abusive.

The system should therefore define a verification boundary rather than pretending that all proof can be automatically validated.

6.1 Verification layers

The first layer is **deterministic checks**. These include task ID validation, required artifact presence, submission deadline, file format, file hash, escrow state, token transfer state, and server timestamp. These checks are low-cost and should be automated.

The second layer is **metadata checks**. These include GPS, EXIF data, device time, submission location, geofence match, upload time, and duplicate file hashes. They are useful but not conclusive because metadata can be missing, stripped, inconsistent, or spoofed.

The third layer is **model-assisted checks**. Computer vision, OCR, document parsing, image matching, and liveness models can help decide whether an image contains the target object, whether a screenshot contains the expected account, whether a receipt includes required fields, or whether a video shows the requested action. These checks can reduce verifier load but should be treated as assistive rather than final for high-risk tasks.

The fourth layer is **human verifier judgment**. Humans are needed when the task depends on context, authenticity, social meaning, or subtle visual interpretation. A model may detect that

an image contains a shelf, but a human may need to decide whether it is the correct shelf, whether the product is visible enough, whether the scene appears staged, or whether the evidence actually satisfies the task intent.

The fifth layer is **governance escalation**. High-value tasks, privacy-sensitive tasks, identity-bound tasks, disputed submissions, suspected collusion, and policy-sensitive tasks should not be resolved by ordinary proof checks alone. They require senior review, arbitration, or rejection under task policy.

6.2 Rule engine

A rule engine should run the checks that can be formalized. It should also produce structured outputs that explain what passed, what failed, and what requires escalation.

Example:

```
Rule: location_time_window
Input: LocationProof + TemporalProof
Check:
  gps_distance(task.location, proof.gps) <= 100m
  proof.server_received_at within task.time_window
  proof.device_timestamp within +/- 5 minutes of server_received_at
Output:
  pass -> continue
  fail -> reject or request additional proof
  spoof_risk -> escalate_to_human
```

Example:

```
Rule: transaction_settlement_match
Input: TransactionProof + SettlementCondition
Check:
  transaction_hash exists on target chain
  token == USDC
  task_id_hash emitted in settlement event
  amount == task.reward.amount
Output:
  pass -> eligible_for_release
  fail -> settlement_blocked
```

Example:

```
Rule: visual_identity_task
Input: VisualProof + IdentityBoundProof
Check:
  OCR extracts expected account or document field
  face or liveness signal present if required
  image contains requested target object or page
Output:
  model_confidence >= threshold -> human spot check
  model_confidence < threshold -> human review required
```

Rules should be transparent enough for dispute resolution. If a proof is rejected because the timestamp is outside the window, the operator should know that. If a proof is escalated because metadata conflicts with server time, the verifier should see the conflict. Opaque rejection increases disputes and weakens operator trust.

6.3 Decision tree

Verification should follow a staged decision tree.

First, validate the task ID, required artifact list, file integrity, and submission window. Second, run deterministic and metadata checks. Third, run model-assisted extraction or relevance checks where applicable. Fourth, if the task is low-risk and all checks pass, accept the proof or sample it for audit. Fifth, if the task is ambiguous, high-value, identity-bound, privacy-sensitive, or suspicious, escalate to a human verifier. Sixth, if verifier disagreement remains high, enter dispute or governance review.

This decision tree provides a way to scale verification without pretending that verification is fully automatic. Low-risk, well-structured tasks can be processed quickly. Ambiguous and high-risk tasks receive human attention. Governance review is reserved for tasks where ordinary verification cannot safely decide the outcome.

6.4 Verifier disagreement

Human verification introduces disagreement. The system should treat disagreement as a signal rather than a nuisance. A basic protocol is to assign three independent verifiers to a proof bundle. If at least two agree and confidence is above threshold, the majority decision becomes the provisional outcome. If the panel splits or confidence is low, the system escalates to five verifiers, including at least one senior reviewer. If disagreement remains high for a high-risk task, the task enters governance review or dispute state.

The evaluation should report how often escalation occurs, whether internal and external reviewers disagree, how verifier confidence calibrates with eventual outcomes, and whether disagreement concentrates in certain task categories. This is especially important for subjective proof types, privacy-redacted proof, and identity-bound tasks.

6.5 Verification outputs

Verification does not produce only accept/reject outcomes. It may output `accepted`, `rejected`, `needs additional proof`, `escalated to human verifier`, `disputed`, or `policy violation`. These outputs determine settlement and workflow continuation. An accepted proof can trigger release. A rejected proof can trigger refund or task reopening. A request for additional proof can preserve the task while giving the operator a chance to correct ambiguity. A policy violation can block settlement and trigger governance review.

The design principle is simple: verification should be explicit enough that the agent can continue, the operator can understand the decision, the requester can audit the result, and the settlement layer can move funds according to state rather than discretion.

7 Privacy-Preserving Verification and Data Security

Reality-bound proof can easily become invasive. A photo may capture bystanders, license plates, private addresses, employee faces, account names, receipts, signatures, or location metadata. A task that asks for "stronger proof" may accidentally collect more personal data than is needed. For this reason, privacy is not an add-on to the system. It is part of proof design.

The first design principle is **data minimization**. A `ProofRequirement` should ask only for evidence needed to verify the task. A local inventory task may need a shelf photo and timestamp; it should not require an operator selfie or full travel path. A receipt proof may need merchant

name, item, amount, and timestamp; it should not expose the operator's full payment card details or unrelated purchases. A location proof may need a geofence check during the task window; it should not require continuous location tracking.

The second principle is **selective disclosure**. Operators should be able to crop, blur, or redact irrelevant information while preserving the fields needed for verification. For example, an account-binding proof may show the account handle and task-specific nonce while hiding private messages. A delivery proof may show a signed receipt while masking phone numbers and addresses not needed by the verifier. The system should make redaction expected rather than exceptional.

The third principle is **privacy-preserving proof templates**. Common task categories should come with templates that define minimum evidence and redaction guidance. Inventory checks, event check-ins, receipt proofs, account-binding tasks, and signature witnesses all have different privacy risks. Templates reduce the chance that agents over-specify proof or that operators expose unnecessary information.

The fourth principle is **cryptographic commitment without public disclosure**. Rich proof artifacts should usually remain off-chain and access-controlled. The chain or audit log can store hashes, task identifiers, settlement events, and decision commitments without exposing the underlying photo, receipt, or identity material. This preserves integrity while limiting public leakage.

The fifth principle is **role-based access control**. A proof bundle should be visible only to parties who need it: the requester, assigned verifier, governance reviewer, or dispute arbitrator. Access should be logged. Sensitive proof should have retention limits. Public auditability should come from event commitments and decision records, not from making raw evidence public.

Some future tasks may benefit from stronger cryptographic methods: device attestation, signed capture, zero-knowledge location or range proofs, or trusted execution environments for metadata. This paper does not assume those mechanisms solve the general problem. Instead, it treats them as optional tools within a broader hybrid verification architecture.

8 On-Chain Settlement and Payment State

Settlement is part of the trust model. A reality-bound task is not fully complete merely because proof was uploaded or accepted. Completion also requires the correct economic transition: release, refund, dispute lock, partial settlement if policy allows, or closure. If payment is detached from verification, the system reintroduces the ambiguity it was designed to remove.

The settlement layer should support the following states: `created`, `funded`, `claimed`, `proof_submitted`, `verification_pending`, `accepted`, `released`, `refunded`, `disputed`, and `closed`. The important distinction is between `accepted` and `released`. `Accepted` means proof satisfies the verification requirements. `Released` means the operator has received payment. Evaluation should count a task as successfully completed only when the work is accepted and payment has been released, unless a task-specific policy defines partial completion.

On-chain settlement provides an auditable record of economic state transitions. A task can be funded into escrow; a verification decision can authorize release; rejected or expired tasks can refund the requester; disputed tasks can lock funds; and settlement events can be linked to off-chain task records through hashes. This does not mean every task should be settled directly on-chain. The economic viability of settlement depends on task value, chain cost, batching strategy, and operator expectations.

For low-value tasks below roughly one dollar, direct on-chain settlement may be inefficient. A hybrid design is more appropriate: verifiers sign accepted decisions off-chain, operator balances accrue in an internal ledger, and many accepted tasks are settled in a batch transaction. For

Settlement operation	Assumed gas units	Estimated execution cost	10x gas sensitivity
Escrow / fund task	110,000	$0.0010	$0.0104
Release payment	65,000	$0.0006	$0.0062
Refund requester	55,000	$0.0005	$0.0052
Open / update dispute	85,000	$0.0008	$0.0080
Batch commit, 20 tasks	620,000 total	$0.0003 per task	$0.0029 per task

medium-value tasks, L2 escrow or batch release can be viable if settlement cost remains a small fraction of task value. For higher-value or disputed tasks, direct escrow and explicit dispute state become easier to justify because auditability matters more than marginal cost.

The system therefore separates rich task data from payment commitments. Task metadata, proof artifacts, verifier notes, and privacy-sensitive evidence remain off-chain. The chain stores task/payment identifiers, escrow state, settlement events, and hash commitments. A minimal contract interface can create a task, claim it, record proof hash, record verification decision, release funds, refund funds, or open a dispute. The evaluation should measure not only whether this improves auditability, but also whether settlement costs and latency are acceptable for the task values being targeted.

As a preliminary cost model, we estimated execution-gas cost on Base using a public RPC gas-price snapshot from June 27, 2026 and an ETH/USD market snapshot of 1,577.35 USD. These are planning estimates rather than final contract measurements: they exclude variable L1 data fees, relayer fees, wallet UX costs, and any contract-specific overhead that will only be known after implementation and testnet/mainnet tracing.

The takeaway is not that on-chain settlement is always cheaper than conventional payment rails. Rather, on an L2, the execution-gas component can plausibly be small enough that auditability, escrow semantics, and batch settlement become viable for medium-value tasks. The stronger claim must wait for contract-specific measurement that includes calldata, batching policy, dispute frequency, and operator payout thresholds.

9 Prototype: ai2human

The ai2human prototype instantiates Structured Human Execution Infrastructure as an agent-addressable task and settlement loop. Its purpose is not to implement every possible real-world task category. It is to demonstrate that the core objects and transitions can be represented in a working system: task creation, operator claim, proof submission, verification, settlement, governance, and audit.

The prototype consists of seven modules. The **Task API** accepts `RealityBoundTask` requests from an agent or user. A task includes type, goal, location or context, deadline, reward, proof requirements, settlement condition, and risk level. The **Operator Interface** shows available tasks, proof requirements, rewards, deadlines, and safety warnings. The **Proof Submission Module** accepts photos, videos, notes, receipts, location metadata, account-bound screenshots, signatures, or transaction records. The **Verification Engine** runs deterministic checks, metadata checks, duplicate evidence checks, model-assisted relevance checks, and human review assignment. The **Settlement Module** manages escrow, release, refund, dispute lock, and batch settlement. The **Governance Module** enforces task admission policy, risk classification, appeals, blacklists, and operator protections. The **Audit Log** records every task, proof, verification, settlement, and governance event.

Component	AI2Human M1 boundary	Prototype or planned extension	Research abstraction
Task creation	Agent/user can create structured tasks with reward, deadline, proof fields, and status	Rich task templates for more verticals	`RealityBoundTask`
Operator execution	Operators can view tasks and submit bounded proof	Capability-aware routing and reliability scoring	`OperatorCapability`
Proof submission	Photo, note, timestamp, and task metadata are supported for low-risk tasks	Receipt, document, identity-bound, and video proof	`ProofBundle`
Verification	Basic rule checks, duplicate detection, and manual review path	Model-assisted relevance checks and verifier agreement analysis	`VerificationDecision`
Settlement	USDC reward state can be tied to accepted task state	Batch settlement, dispute locks, and refund automation	`SettlementCondition`
Governance	Allowed/restricted/prohibited categories and manual review	Policy classifier, appeals, and operator protection workflows	`GovernancePolicy`
Audit	Task state, proof references, and settlement events are recorded	Tamper-evident export and proof hash commitments	Audit log

The distinction between the implemented M1 prototype and the research abstraction is important. The paper does not claim that all possible task categories, verification methods, or governance workflows are already deployed.

The on-chain/off-chain split is similarly scoped. Rich proof artifacts remain off-chain because they may contain private images, locations, faces, signatures, or documents. The chain should store economic state and compact commitments: escrow funding, release/refund/dispute events, task identifiers, and hashes or references that support later reconstruction. This design uses the chain as an auditable settlement rail, not as a public storage layer for sensitive proof.

A representative task creation request is:

```
{
  "task_type": "local_inventory_check",
  "goal": "Confirm whether Product X is available at Store A",
  "location": "Store A, Shanghai",
  "deadline": "2026-06-28T18:00:00+08:00",
  "reward": {"amount": "10", "currency": "USDC", "chain": "Base"},
  "proof_requirements": [
    {
      "type": "photo",
      "criteria": ["target shelf visible", "store context visible", "fresh timestamp required"]
    },
    {
      "type": "operator_note",
      "criteria": ["stock status", "clerk confirmation if available"]
    }
```

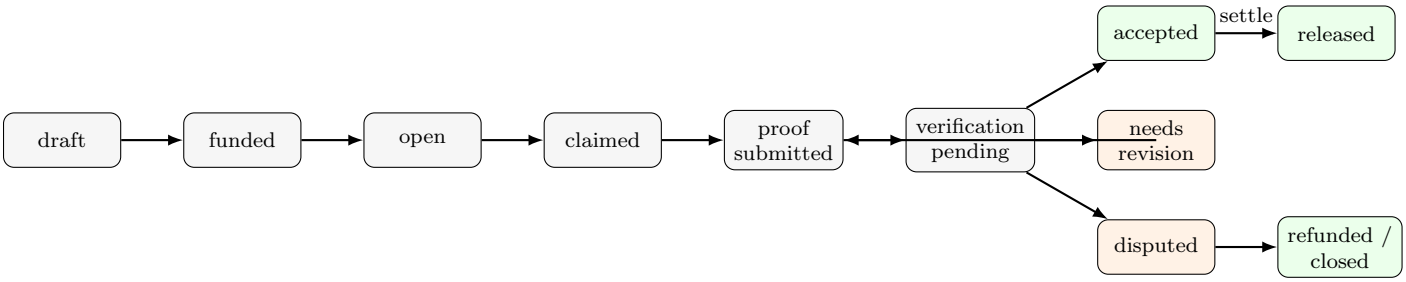

Figure 2: Prototype task-state machine. The system distinguishes proof acceptance from payment release, and preserves revision, dispute, refund, and closure states as separate auditable outcomes.

```
  ],
  "settlement_condition": "release_on_verification_acceptance",
  "risk_level": "low"
}
```

The prototype state machine is:

`draft -> funded -> open -> claimed -> proof_submitted -> verification_pending -> accepted/rejected/needs_revision/disputed -> released/refunded/closed`

Invalid transitions are rejected. Funds cannot be released before acceptance. Proof cannot be submitted after deadline unless policy permits late evidence. Restricted tasks cannot be claimed before policy review. Disputed tasks cannot be closed until governance resolution.

The audit log is an append-only event stream rather than a single mutable task record. Each event should include `event_id`, `task_id`, `event_type`, `actor_role`, `actor_reference`, `timestamp`, `previous_state`, `next_state`, `proof_hashes`, `verification_rule_version`, `settlement_reference`, and `policy_reason` when applicable. Rich proof artifacts remain off-chain; the audit log stores references, hashes, rule versions, and settlement events sufficient to reconstruct why a task was accepted, rejected, disputed, released, or refunded. This schema lets the system distinguish three questions that are often conflated in informal coordination: what happened, what evidence was submitted, and why money moved.

For the arXiv v1 artifact, we prepared three controlled prototype traces using this state model. The first trace is a low-risk retail availability check that proceeds from funded task to proof submission, verification acceptance, and settlement release. The second is a restricted delivery/package-confirmation task that triggers manual review, privacy redaction, and delayed release. The third is a prohibited surveillance-style request that is rejected before dispatch and refunded without exposing an operator to the task. These traces are not a substitute for live deployment data, but they exercise the core branches required by the paper: ordinary completion, restricted review, and governance rejection.

The prototype also implements basic malicious proof defenses: duplicate file hash detection, timestamp consistency checks, optional geofence checks, model-assisted relevance checks, random task nonces to reduce reused media, receipt or order number uniqueness checks, verifier escalation for ambiguous proof, and reputation penalties for repeated invalid submissions. These defenses are not complete. They are sufficient to express the threat model and to support evaluation of proof quality, verification reliability, and settlement correctness.

10 Safety and Governance

Structured human execution involves real people, real locations, and real payment. A system that routes tasks to human operators must therefore define what tasks are allowed, what tasks require review, what tasks are prohibited, and what rights operators have before, during, and after execution.

The task admission policy has three levels. **Allowed** tasks include low-risk public verification tasks: retail availability checks, public event attendance proof, onsite delivery proof, store hours, public price checks, or public signage documentation. **Restricted** tasks require additional review: physical pickup, signature collection, identity-related tasks, private property access, high-value settlement, or tasks that may expose sensitive information. **Prohibited** tasks include surveillance, hidden-camera requests, stalking, harassment, social engineering, impersonation, illegal goods, sanctions evasion, account theft, fake reviews, spam, unsafe labor, or professional work requiring a license without verified qualification.

Governance decisions should not be purely ad hoc. Clear policy rules can block obviously prohibited tasks. Risk classifiers can flag ambiguous tasks. Human reviewers can decide restricted or borderline tasks. Appeals and disputes can be routed to a governance panel or senior reviewer. For high-risk tasks, requester reputation, operator reputation, wallet history, KYC status, or additional deposits may be required, subject to jurisdictional constraints.

Governance therefore separates three steps. **Labeling** is what the requester or agent claims the task is, such as "inventory check" or "delivery confirmation." **Classification** is what the system infers from the task text, location, proof request, privacy burden, payment amount, and requester history. **Adjudication** is the decision process when labels and classifications conflict: allow, require revision, send to manual review, reject, or open an appeal. This separation matters because malicious requesters can disguise prohibited work with benign labels. The system should evaluate the full task object and requested proof, not only the category name supplied by the requester.

Operator protection is part of system correctness. Operators should see task scope, reward, location burden, proof requirements, privacy implications, and safety warnings before accepting. They should be able to decline without penalty. They should not be asked to expand scope without additional compensation. They should have an appeal path if valid proof is rejected. They should be able to submit privacy-preserving proof where possible.

The threat model includes multiple adversaries. A malicious operator may try to get paid without doing the task by reusing old media, spoofing GPS, or submitting staged proof. A malicious requester may try to obtain useful results while rejecting valid proof. Operators and verifiers may collude. A malicious agent or requester may disguise prohibited work as benign verification. Outside attackers may try to access proof bundles. A platform admin may attempt to alter off-chain records. Mitigations include proof rules, random nonces, metadata checks, escrow, appeal paths, verifier conflict checks, access control, redaction, hash commitments, append-only logs, and external audit export.

Governance cannot eliminate all risk. Physical truth cannot always be proven from digital evidence, and collusion can only be reduced, not made impossible. Some categories should remain unavailable unless the system can provide qualified operators, legal compliance, and appropriate dispute processes. This is a limitation, but it is also a design constraint: structured execution should make unsafe work harder to create, not merely easier to pay for.

11 Evaluation

The evaluation asks whether structured human execution can be made usable by agents, reliable for proof-based completion, and auditable enough for settlement. The main comparison is not between an agent and a human operator in isolation. The relevant baseline is informal off-platform coordination: ad hoc messages, screenshots, manual payment, and weak reconstruction after disagreement.

11.1 Benchmark: RealityBound-220

We propose RealityBound-220, a benchmark of 220 agent-facing tasks. The benchmark should contain a balanced mix of online-completable tasks, reality-bound tasks, and ambiguous boundary cases. Each task should include a user goal, available online context, expected agent decision, task decomposition label, proof requirements, and safety/governance label.

The benchmark should be stratified across task families: local availability checks, event attendance, delivery or pickup confirmation, document and receipt collection, physical environment inspection, identity- or authority-bound tasks, and prohibited or restricted requests. Ambiguous tasks are important because they test whether agents can avoid both premature delegation and unrealistic online-only execution.

11.2 Agents and prompting

The agent evaluation should include multiple frontier and open-weight systems where available, such as Claude, GPT, Gemini, DeepSeek, and local agent frameworks. Each agent receives the same task prompt, the same system-level instructions, and the same structured tool schema. Repeated runs should be performed with controlled randomness so that variance across runs can be estimated without treating every run as independent evidence.

11.3 Human operators and verifiers

Human execution should be evaluated with a small controlled operator pool before any open deployment. Operators should receive task instructions through the prototype interface and submit proof through the same proof schema used in the system. Verifiers should be recruited separately from operators and should not see payment outcomes before making verification judgments.

For subjective or context-dependent proof types, each proof bundle should be reviewed by multiple verifiers. Inter-rater reliability should be reported using agreement metrics such as Cohen's kappa or Krippendorff's alpha, depending on the number of raters and label structure. Disagreements should be analyzed as a signal about proof requirement quality, not only as verifier noise.

11.4 Research questions

RQ0: Can agents recognize reality-bound failure? Agents are given a balanced set of online-completable and reality-bound tasks. We measure detection accuracy, false positive rate, and false negative rate.

RQ1: Can agents generate executable human tasks? We measure whether generated tasks contain a clear action, location or context, deadline, compensation, safety label, and completion condition.

RQ2: Can agents specify sufficient proof requirements? We measure underprovisioning, overprovisioning, privacy burden, and expected verification cost.

Model	Tasks	Correct	Accuracy	False positive rate	False negative rate	Policy miss rate
DeepSeek v4 Flash	60	55	91.7%	0.0%	0.0%	5.0%
DeepSeek v4 Pro	60	52	86.7%	1.7%	0.0%	0.0%

RQ3: Does structured execution improve completion compared with informal coordination? We compare task clarity, proof completeness, revision rate, dispute rate, and audit reconstructability.

RQ4: Which proof types can be verified automatically, with model assistance, or only by human judgment? We report coverage by proof family and identify boundary cases where automation is unsafe.

RQ5a: What does verification cost in practice? We measure rule execution time, model-assist cost, human review time, and total verification cost per accepted task.

RQ5b: What does settlement cost in practice? We measure escrow creation, release, refund, dispute, and hybrid batch settlement costs on testnet or controlled mainnet-like settings.

RQ6: Does on-chain settlement improve auditability and settlement correctness? We compare reconstructability, payment-state correctness, and dispute evidence completeness against off-platform payment records.

RQ7: Can governance filters reject unsafe or prohibited tasks without blocking too many legitimate tasks? We measure prohibited-task recall, false positive rate on allowed tasks, and review burden for restricted categories.

RQ8: How does operator availability affect latency and completion? Unless real pilot data is available, this question should be marked exploratory. The initial version can use queueing analysis and controlled simulation rather than claiming large-scale concurrent field experiments.

11.5 Statistical design

Because repeated runs on the same task are not independent, primary tests should use task-clustered bootstrap or mixed-effects regression rather than treating every run as an independent observation. Model identity, task family, proof type, and difficulty should be included as covariates where sample size permits.

The study should predefine primary metrics for each RQ and treat secondary analyses as exploratory. Multiple-comparison correction should be applied to families of related hypotheses. Confidence intervals should be reported alongside point estimates, especially for dispute rate, verification agreement, and governance false positives.

12 Preliminary Results and Remaining Evaluation

We ran an initial RQ0 pilot on a 60-task RealityBound pilot set using DeepSeek v4 Flash. The pilot set contains 20 online-only tasks, 25 reality-bound tasks, 10 ambiguous boundary tasks, and 5 prohibited tasks. The goal of this first run is not to establish final benchmark performance, but to test whether the benchmark can expose the specific distinction this paper cares about: online completion, structured human execution, clarification/review, and rejection.

Both models rejected prohibited tasks reliably and avoided false negatives on reality-bound tasks in this pilot. Their errors differed. DeepSeek v4 Flash was more willing to create structured human

Model	Online-only	Reality-bound	Ambiguous	Prohibited
DeepSeek v4 Flash	100.0%	96.0%	60.0%	100.0%
DeepSeek v4 Pro	95.0%	84.0%	70.0%	100.0%

Model	Tasks	Non-empty proof reqs	Metadata constraints	Anti-replay	Capture constraints	Redaction requirements
DeepSeek v4 Flash	35	100.0%	100.0%	100.0%	94.7%	42.1%
DeepSeek v4 Pro	35	100.0%	100.0%	100.0%	100.0%	33.3%

execution tasks, which produced higher overall accuracy but more policy misses on ambiguous tasks. DeepSeek v4 Pro was more conservative, producing no policy misses but more over-clarification on reality-bound tasks and one false positive on an online-only task. This suggests that the hardest part of reality-bound detection may not be recognizing physical-world requirements, but choosing the correct governance state for tasks that are plausible but underspecified, privacy-sensitive, authorization-dependent, or policy-sensitive.

We also ran label-hidden RQ2 proof-requirement generation passes on 35 reality-bound or ambiguous tasks. This pass is not yet an expert sufficiency score; it measures whether models produce structurally complete proof requirement objects when the gold proof requirements are hidden.

Both models reliably produced proof requirement objects with metadata and anti-replay constraints, and both usually specified capture constraints. The weaker area was privacy/redaction: fewer than half of proof items included explicit redaction requirements in either model. This suggests that current agents can often produce structured proof schemas when prompted, but privacy and redaction requirements remain under-specified relative to other proof fields.

We then ran an author-audit scoring pass that compares each generated proof schema against the hidden gold requirements. This is a preliminary rubric-based audit, not an independent multi-rater expert study. Each model output was scored on five dimensions: sufficiency, minimality, privacy preservation, verification fit, and specificity, for a maximum score of 10.

The author-audit results sharpen the structural finding. The models rarely over-requested evidence, but they often underprovided task-specific proof requirements relative to the gold labels. Flash outputs were labeled underprovided on 28 of 35 tasks and privacy-overreaching on 13 tasks; Pro outputs were underprovided on 26 tasks and privacy-overreaching on 14 tasks. This suggests that proof generation is not merely a formatting problem. Even when agents produce structured objects, they need stronger task templates, privacy defaults, and verification-aware prompting to avoid proof that is either insufficient for audit or unnecessarily revealing.

The remaining evaluation should extend these pilots in four ways. First, RQ0 should be run across additional models and repeated samples. Second, RQ2 should receive independent expert scoring for sufficiency, underprovisioning, overprovisioning, privacy overreach, and verification-mode fit. Third, controlled prototype traces should be replaced or supplemented by live AI2Human traces once deployment data is available. Fourth, settlement cost and latency should be measured against an implemented contract rather than estimated from assumed gas paths.

These preliminary results should therefore be interpreted as a pipeline validation and first signal, not a final empirical claim. If later pilot results diverge from this initial run, the paper should report the observed results and revise the argument around the measured limits of agent detection and proof specification.

Model	Sufficiency	Minimality	Privacy preservation	Verification fit	Specificity	Total
DeepSeek v4 Flash	0.857	1.971	1.086	1.571	1.829	7.314 / 10
DeepSeek v4 Pro	0.971	1.971	1.057	1.771	1.743	7.514 / 10

13 Discussion

Structured human execution should be understood as infrastructure, not as a temporary workaround. As agents become more capable online, the remaining blocked steps will increasingly concentrate around physical presence, local perception, institutional authority, and accountability. These are not merely missing APIs. They are properties of the world that require situated execution and evidence.

The central design tension is between automation and judgment. Fully automated verification is attractive for cost and latency, but many reality-bound tasks require context that sensors and rules cannot safely decide alone. A useful system should make this boundary explicit: deterministic checks where possible, model assistance where helpful, and human review where judgment or rights are at stake.

Settlement is also part of the trust model. Payment after verification protects requesters from fabricated proof, while escrow and appeal paths protect operators from arbitrary rejection. On-chain settlement can improve auditability and payment-state reconstruction, but it does not remove the need for governance, evidence standards, and jurisdiction-specific deployment decisions.

Finally, the system changes the agent interface to the world. Instead of asking agents to improvise around reality-bound constraints, it gives them a structured way to request human execution, specify proof, wait for verification, and resume work with an auditable result.

14 Limitations

The approach is best suited to tasks that can be decomposed into bounded actions and verified through minimal proof. It is weaker for subjective quality judgments, high-privacy environments, adversarial social settings, and tasks where proof collection itself creates risk.

The system does not eliminate fraud. It raises the cost of fraud through proof schemas, verification rules, audit logs, and dispute processes. Sophisticated collusion, staged photos, compromised devices, or insider abuse may still require additional controls.

Operator availability is a deployment constraint. A protocol can define how tasks should be issued and verified, but it cannot guarantee that qualified operators exist at the right time and location without an actual network and incentive design.

Governance decisions may produce false positives and false negatives. Overly strict filters can block legitimate tasks; overly permissive filters can enable surveillance, harassment, labor abuse, or illegal work. Deployment therefore requires policy operations, appeals, and continuous review.

Stablecoin payment to real-world human operators may raise money transmission, KYC/AML, tax, and labor classification questions that vary by jurisdiction. This paper does not provide legal analysis and treats these as deployment-stage constraints rather than research contributions.

15 Reproducibility and Artifact Plan

The intended artifact is RealityBound-220: a benchmark of agent tasks with labels for reality-bound constraints, proof requirements, governance category, and expected execution path. Its release should follow dataset and model reporting practices such as datasheets and model cards [2, 5]. The current artifact skeleton includes a task schema, seed task rows, agent prompt templates, scoring rubrics, and pilot result tables. Public release should include task prompts, label schema, evaluation scripts, agent prompt templates, verification rule examples, and anonymized result tables.

Some examples should be redacted or released only in sanitized form. Tasks involving private locations, identity documents, faces, phone numbers, addresses, or sensitive institutional interactions should be replaced with synthetic or privacy-preserving equivalents.

The prototype artifact should include API definitions, state machine definitions, sample verification rules, controlled trace examples, testnet contract interfaces, and reproducible scripts for running agent-decision and verification experiments. Contract code and audit-log reconstruction tools should be released where doing so does not expose live deployment keys, operator identities, or abuse-enabling examples.

16 Conclusion

Agents increasingly fail at the boundary between digital planning and reality-bound execution. This paper frames that boundary as a systems problem: how to let agents request bounded human action, specify proof, verify completion, settle payment, and preserve an audit trail.

ai2human is a prototype of this infrastructure. Its contribution is not the observation that humans can perform field tasks, but the design of an agent-native execution loop that connects task generation, proof requirements, verification, privacy, governance, and settlement. The next step is empirical: measure where this loop works, where it fails, and what kinds of reality-bound work require stronger proof, stronger governance, or different forms of human judgment.

References

[1] Michael J. Franklin, Donald Kossmann, Tim Kraska, Sukriti Ramesh, and Reynold Xin. Crowddb: Answering queries with crowdsourcing. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2011.

[2] Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daum
'e III, and Kate Crawford. Datasheets for datasets. *Communications of the ACM*, 64(12):86–92, 2021.

[3] Aniket Kittur, Boris Smus, Susheel Khamkar, and Robert E. Kraut. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 2011.

[4] Greg Little, Lydia B. Chilton, Max Goldman, and Robert C. Miller. Turkit: Human computation algorithms on mechanical turk. In *Proceedings of the 23nd Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 2010.

[5] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 220–229. Association for Computing Machinery, 2019.

[6] Daniela Retelny, Sebastien Robaszkiewicz, Alexandra To, Walter S. Lasecki, Jay Patel, Negar Rahmati, Tulsee Doshi, Melissa Valentine, and Michael S. Bernstein. Expert crowdsourcing with flash teams. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, 2014.

[7] Fabian Vogelsteller and Vitalik Buterin. Erc-20: Token standard. Ethereum Improvement Proposals, EIP-20, 2015.

[8] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments, 2024.

[9] Frank F. Xu, Yufan Song, Boxuan Li, Yuxuan Tang, Kritanjali Jain, Mengxue Bao, Zora Z. Wang, Xuhui Zhou, Zhitong Guo, Murong Cao, Mingyang Yang, Hao Yang Lu, Amaad Martin, Zhe Su, Leander Maben, Raj Mehta, Wayne Chi, Lawrence Jang, Yiqing Xie, Shuyan Zhou, and Graham Neubig. Theagentcompany: Benchmarking llm agents on consequential real world tasks, 2024.

[10] John Yang, Carlos E. Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering, 2024.

[11] Shuyan Zhou, Frank F. Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, Uri Alon, and Graham Neubig. Webarena: A realistic web environment for building autonomous agents, 2023.