# PROVISIONAL PATENT APPLICATION SPECIFICATION

## Title of the Invention

DETERMINISTIC COMBAT SPORTS OFFICIATING SYSTEM WITH CRYPTOGRAPHIC SETTLEMENT PROOF, SCHEMA-CONSTRAINED VISION TELEMETRY, NON-DESTRUCTIVE CORRECTION OVERLAYS, MULTI-AGENT PARALLEL SYNTHESIS PIPELINE, AUTOMATED TALENT ACQUISITION ENGINE, AND CRYPTOGRAPHICALLY SECURED REAL-TIME IN-PLAY SPORTS WAGERING SETTLEMENT

## Inventor

Dustin Blaine Salinas, Las Vegas, Nevada, United States

## Applicant

Dustin Blaine Salinas, filing pro se in personal capacity

## Divisibility Notice

This provisional application expressly discloses multiple related inventive embodiments. Applicant reserves the right to file separate non-provisional applications claiming priority to this provisional filing date, each directed to distinct inventive subject matter described herein.

## PART 0 - CROSS-REFERENCE AND INCORPORATION

This application incorporates by reference the following prior-authored technical documents, each fully describing aspects of the inventions claimed herein, authored by Dustin Blaine Salinas: (1) *Provisional Patent: Deterministic Asymmetric Ruleset Enforcement via Schema-Constrained Boundaries*; (2) *Patent Specification: Cryptographically Signed Sports Betting Settlement Proofs*; (3) *White Paper: The Correction Overlay Method - Securing Integrity in Professional Grappling via Cryptographic Rubric Registries*; (4) *Provisional Patent Application: Multi-Agent Parallel Sports Vision Orchestration and Synthesis Pipeline*; (5) *Patent Application Strategy for Grappling Innovations*; (6) *PGF Scout: The Engine & The Empire - Technical Architecture & Commercial Integration for the Franchise Era*; (7) *PGF Franchise Monetization: Technical Architecture & Commercial Integration*; and (8) *Absolute Ground Truth: Technical Documentation of the PGF Scoring System*.

---

## BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a three-tier S.C.O.U.T. processing architecture in which parallel grading agents generate telemetry, a verification tier validates schema and RubricEpoch conditions, and a synthesis tier produces scoring, fantasy, franchise, and sportsbook outputs while preserving isolation boundaries between raw frames, rubric values, and downstream synthesis.

FIG. 2 is a flowchart illustrating a SettlementProof four-layer hash-binding process in which event metadata, video evidence bytes, active model state, and regulatory ruleset context are separately hashed, combined into a single root hash, timestamped, signed, and distributed to sportsbook, regulatory, and anchoring endpoints.

FIG. 3 is a diagram of a RubricEpoch chaining protocol in which genesis and subsequent ruleset epochs are stored in an append-only collection, validated from genesis to active epoch before scoring or settlement, and subject to a fallback process that flags proofs generated under database-unavailable conditions.

FIG. 4 is a flowchart of a non-destructive correction overlay process in which an original AI-reviewed classification is preserved, conflicts are routed through authority and integrity checks, and any human override is appended as a separate overlay record for point-in-time corrected querying without deleting the original record.

FIG. 5 is a system diagram of a Draft Yourself talent-data flywheel in which user-submitted match video, explicit consent, S.C.O.U.T. processing, fighter-card generation, fantasy distribution, top-quintile invitation logic, and append-only talent-ledger history produce a compounding training corpus and recruiting ledger.

FIG. 6 is a real-time in-play sportsbook settlement flow diagram in which raw video feeds are processed by S.C.O.U.T. agents, converted into schema-constrained market ticks, adjudicated by ContextOS, filtered by trust state, bound into a SettlementProof envelope, signed with ECDSA-P256, and delivered to sportsbook and regulatory endpoints for deterministic in-play market settlement.

*via Cryptographic Rubric Registries*; (4) *Provisional Patent Application: Multi-Agent Parallel Sports Vision Orchestration and Synthesis Pipeline*; (5) *Patent Application Strategy for Grappling Innovations*; (6) *PGF Scout: The Engine & The Empire — Technical Architecture & Commercial Integration for the Franchise Era*; (7) *PGF Franchise Monetization: Technical Architecture & Commercial Integration*; and (8) *Absolute Ground Truth: Technical Documentation of the PGF Scoring System*.

## PART 0A — BACKGROUND OF THE INVENTION (SHARED)

### Field of the Invention

The present inventions reside at the intersection of computer vision systems, distributed multi-agent artificial intelligence architectures, cryptographic ledger systems, real-time sports data settlement infrastructure, and automated athletic scouting platforms. More specifically, the inventions address the transformation of high-velocity, submission-based grappling competition data into deterministic, cryptographically verifiable, settlement-grade financial instruments suitable for regulated sports wagering, franchise equity management, and talent acquisition.

### Description of the Related Art and Problem Statement

The professional grappling industry, represented by leagues such as the Professional Grappling Federation (PGF), operates under a set of structural data challenges that prevent integration with institutional wagering markets and franchise-grade capital structures.

**Prior Art Landscape and Deficiencies:**

Traditional sports officiating systems rely on human judges whose determinations are non-replicable and subjective. U.S. Patent No. 7,005,970 B2 (Intel Corporation) discloses a computer-implemented officiating system for sports; however, it does not disclose decoupling visual classification from mutable scoring logic through a schema-constrained interface, nor does it disclose cryptographic chaining of ruleset epochs. U.S. Patent Application No. 20170364753 A1 (IBM) discloses a referee assistance system utilizing gaze tracking; however, it relies on human-in-the-loop confirmation and does not disclose an append-only immutable epoch persistence layer or a non-destructive correction overlay methodology. U.S. Patent Application No. 20200289887 A1 (Simpson) discloses sensor-based officiating for boundary-defined sports; however, it does not disclose vision-based telemetry validation against a closed position registry, cryptographic multi-layer settlement proofs, or schema-constrained hallucination elimination. No prior art discloses the combination of: (i) a Pydantic v2-enforced closed-registry interface layer, (ii) a four-layer SHA-256 SettlementProof envelope binding video evidence to AI model state to versioned ruleset epoch, (iii) non-destructive overlay-based correction methodology with forensic audit preservation, (iv) a three-tier multi-agent Grading/Verification/Synthesis pipeline with fail-fast chain integrity validation, and (v) a top-quintile automated talent identification engine generating standardized cryptographically-backed athlete research profiles.

**Commercial Problem Statement:**

The global sports betting market exceeded $330 billion in legal wagers across 39 states following the Professional and Amateur Sports Protection Act (PASPA) repeal. Institutional sportsbooks require mathematical, non-repudiable settlement proofs before approving in-play wagering on a sport. Traditional grappling promotions are "toxic assets" for betting operators because scoring relies on non-deterministic human judgment with no verifiable audit trail. A 71-minute match (e.g., the Season 1 Krikorian v. Bacallao encounter) is structurally un-bettable due to temporal unpredictability and outcome opacity. The PGF's 6-minute sprint and 10-minute EBI finale format solves the temporal problem; however, a cryptographic data integrity layer is required to solve the oracle problem. The present inventions collectively provide this layer.

## PART I — INVENTION TARGET 1

## DETERMINISTIC ASYMMETRIC RULESET ENFORCEMENT VIA SCHEMA-CONSTRAINED BOUNDARIES ("SALINAS ARCHITECTURE")

### I.1 Technical Field

This invention relates to a computer-implemented system for automated sports officiating that transforms probabilistic, non-deterministic outputs of computer vision (CV) and vision-language model (VLM) agents into deterministic, legally defensible scoring outcomes by enforcing a schema-constrained interface layer between a perception tier and a logic tier.

### I.2 Summary of the Invention

The core invention is a **decoupled officiating architecture** comprising two structurally isolated processing layers: (1) a Constrained Interface Layer that forces visual telemetry into a closed, validated schema, and (2) an Asymmetric Scoring Engine that applies mutable, league-specific point weights to validated tokens without modifying or retraining the underlying vision system. The architectural separation creates a technical "firewall" between physical event classification and financial settlement logic, enabling: (a) hallucination elimination through hard schema boundaries; (b) instant ruleset modification without model retraining; and (c) multi-league support from a single vision installation.

### I.3 Detailed Technical Description

#### I.3.1 Layer 1 — The Constrained Interface (Hallucination Elimination Gate)

The Interface Layer receives raw probabilistic output from one or more AI vision agents and enforces structural determinism through the following mechanism:

**Pydantic v2 Enforcement Model:**

```
from pydantic import BaseModel, Field, validator
from typing import Literal, Dict, Any, Optional
from enum import Enum
```

```
class SubmissionTier(str, Enum):
    KILL = "kill"        # Choke submissions — 6 point base value
    BREAK = "break"      # Joint lock submissions — 3 point base value

class PositionEnum(str, Enum):
    # CLOSED REGISTRY — 160+ entries (partial listing for claim scope)
    GUARD_PASS = "guard_pass"
    MOUNT_ATTAINED = "mount_attained"
    BACK_TAKE = "back_take"
    RNC_CHOKE = "rnc_choke"
    TRIANGLE_CHOKE = "triangle_choke"
    ARMBAR = "armbar"
    HEEL_HOOK_INSIDE = "heel_hook_inside"
    HEEL_HOOK_OUTSIDE = "heel_hook_outside"
    KNEEBAR = "kneebar"
    TOEHOLD = "toehold"
    ESTIMA_LOCK = "estima_lock"
    WRIST_LOCK = "wrist_lock"
    KIMURA = "kimura"
    AMERICANA = "americana"
    DARCE_CHOKE = "darce_choke"         # D'Arce — enforced exact snake_case
    ANACONDA_CHOKE = "anaconda_choke"
    GUILLOTINE_FRONT = "guillotine_front"
    BOW_AND_ARROW = "bow_and_arrow"
    OMOPLATA = "omoplata"
    TWISTER = "twister"                  # Twister lock — sub_under_60s restriction a
    BUTTERFLY_GUARD = "butterfly_guard"
    HALF_GUARD = "half_guard"
    RUBBER_GUARD = "rubber_guard"
    THROW_BY = "throw_by"
    DOUBLE_LEG = "double_leg"
    SINGLE_LEG = "single_leg"
    LATERAL_DROP = "lateral_drop"
    NEUTRAL_STANDING = "neutral_standing"
    NEUTRAL_GROUND = "neutral_ground"
    STALL_DETECTED = "stall_detected"    # Triggers 20-second shot clock
    # ... [160+ total registry members — Exhibit A provides complete enumeration]

class GrapplingTelemetry(BaseModel):
    event_id: str                        # UUID, system-generated
    match_id: str                        # Foreign key to match record
    athlete_id: str                      # Fighter identifier
    transition_name: PositionEnum        # MUST match closed registry
    submission_tier: Optional[SubmissionTier] = None
    timestamp: float                     # Epoch seconds, microsecond precision
    t_sec: float                         # Match-elapsed time in seconds
    confidence_score: float = Field(ge=0.0, le=1.0)
    sub_under_60s_count: int = Field(ge=0, le=1)  # Elbow Genie bonus trigger
    p_finish: float = Field(ge=0.0, le=1.0)       # Kinematic finish probability
    positional_dominance_pct: float = Field(ge=0.0, le=100.0)
    metadata: Dict[str, Any]
    agent_consensus_count: int = Field(ge=2)  # Minimum 2 of 3 agents required

    @validator('transition_name')
    def validate_registry_membership(cls, v):
```

```
        if v not in PositionEnum.__members__.values():
            raise ValueError(f"REGISTRY_REJECTION: '{v}' is not in the Closed Positio
        return v

    @validator('sub_under_60s_count')
    def validate_elbow_genie(cls, v, values):
        if v == 1 and values.get('t_sec', 999) >= 60.0:
            raise ValueError("TEMPORAL_CONFLICT: sub_under_60s_count=1 requires t_se
        return v
```

**Closed Position Registry:** The system implements a Closed Position Registry comprising at minimum 160 visual grappling transitions and positional states. This registry constitutes the complete "universe" of observable events. Any vision agent output not contained within this registry is rejected with a `REGISTRY_REJECTION` exception — the transition is discarded, not passed to the scoring layer. This is the hallucination elimination mechanism.

**Consensus Mechanism:** The system orchestrates a minimum of three (3) parallel vision agents. A telemetry packet is passed to the Interface Layer only after a majority consensus (minimum 2 of 3 agents identifying the same `transition_name`) is reached. The `agent_consensus_count` field records the consensus value for audit.

### I.3.2 Layer 2 — The Asymmetric Scoring Engine

The Scoring Engine receives validated `GrapplingTelemetry` packets and applies the active ruleset from the current `RubricEpoch` to produce financial settlement values:

**Lethality Heuristic (Mathematical Enforcement):**

- `BREAK` (joint locks: armbar, heel_hook, kneebar, etc.): base value = **3.0 points**
- `KILL` (chokes: RNC, triangle, D'Arce, anaconda, etc.): base value = **6.0 points**
- Kill weight scalar = exactly **2.0× Break scalar** — enforced as a mathematical constant, not configurable
- `Elbow Genie` bonus: `sub_under_60s_count == 1` → `+1.0 point` (temporal bonus, configurable)
- `Stall penalty`: `stall_detected` event → triggers 20-second shot clock; second offense = point deduction per active ruleset

**Hot-Swappable Rubric Interface:**

```
class ScoringEngine:
    def __init__(self, rubric_epoch: RubricEpoch):
        self.rubric = rubric_epoch
        self._validate_lethality_constant()

    def _validate_lethality_constant(self):
        # INVARIANT: Kill weight must always be exactly 2.0x Break weight
        assert self.rubric.kill_weight == self.rubric.break_weight * 2.0, \
            "INVARIANT_VIOLATION: Lethality constant 2.0x must be preserved"

    def score(self, telemetry: GrapplingTelemetry) -> float:
```

```
        base = self.rubric.weights.get(telemetry.transition_name, 0.0)
        bonus = 1.0 if telemetry.sub_under_60s_count == 1 else 0.0
        return base + bonus
```

The scoring rubric file is an external, mutable configuration. Point values (other than the Kill:Break 2:1 invariant) can be modified by loading a new `RubricEpoch`, which triggers epoch sequencing in the persistence layer without any modification to vision model weights or the Interface Layer.

### I.3.3 Specific Named Enforcement Rules

The following rules are specifically enforced at the schema level and cannot be overridden by vision agent output:

1. **D'Arce Lexical Rule:** The D'Arce choke is enforced as `"darce_choke"` (no apostrophe, snake_case). Any variation (`"D'Arce"`, `"darce"`, `"D_arce_choke"`) is rejected by the validator as a non-registry string.
2. **Twister Restriction:** `transition_name == "twister"` is only admitted to the submission tier if `sub_under_60s_count == 1` AND a minimum of 3 of 3 agent consensus is recorded. A Twister with 2/3 consensus is flagged for human review, not auto-scored.
3. **Sub-Under-60 Temporal Lock:** `sub_under_60s_count = 1` requires `t_sec < 60.0`, enforced at the Pydantic validator level with a `TEMPORAL_CONFLICT` exception.

### I.4 Provisional Independent Claims

**Claim I-1 (Decoupled Officiating Method):** A computer-implemented method for automated sports officiating comprising: receiving, by a constrained interface layer, probabilistic output from at least one AI vision agent; validating said output against a closed registry of string literal enumeration values using a schema validation library; rejecting any vision agent output not contained within said closed registry; passing only validated telemetry packets to a scoring engine; wherein said scoring engine applies point weights from an external, mutable ruleset file without modification to said AI vision agent or said constrained interface layer.

**Claim I-2 (Consensus + Schema Enforcement):** A method for enforcing determinism in non-deterministic AI vision agents comprising: orchestrating at least three AI vision agents in parallel against a common video input; requiring majority consensus among said agents on a transition identifier before admitting said identifier to a schema validation step; enforcing validation through a Pydantic v2 data model constraining output to a predefined closed registry of snake_case string literal enumeration values; rejecting non-conforming output at the interface layer boundary.

**Claim I-3 (Asymmetric Lethality Constant):** A scoring engine for combat sports officiating comprising: a weighted lookup table mapping validated visual telemetry tokens to floating-point score values; wherein submissions classified as choke-based receive a base score value that is exactly 2.0 times the base score value of submissions classified as joint-lock-based; wherein said 2.0 multiplier is enforced as a mathematical invariant that cannot be modified by rubric configuration files.

**Claim I-4 (Closed Position Registry as Hallucination Gate):** A functional component of a vision interface comprising: a predefined registry of at least 160 grappling-specific transition identifiers expressed as snake_case string literals; wherein any AI vision agent output not matching a registry member triggers a programmatic rejection exception; wherein rejected outputs are logged but not passed to the scoring tier.

**Claim I-5 (Hot-Swappable Epoch Rubric):** A method for managing ruleset drift in automated sports officiating comprising: storing scoring weights in an append-only persistence layer comprising sequenced rubric epoch records; applying said weights from the current active epoch to validated telemetry; enabling modification of scoring weights by creating a new epoch record without modification to underlying neural network weights or schema validation logic.

## PART II — INVENTION TARGET 2

## CRYPTOGRAPHICALLY SIGNED SPORTS BETTING SETTLEMENT PROOFS ("SETTLEMENTPROOF ENVELOPE")

### II.1 Technical Field

This invention relates to a cryptographic infrastructure for the automated settlement of sports wagering markets in high-frequency, submission-based combat sports environments, specifically a four-layer simultaneous SHA-256 binding mechanism that produces a mathematically non-repudiable settlement proof eliminating temporal fraud ("past-posting") and oracle dependency.

### II.2 Summary of the Invention

The **SettlementProof Envelope** is a unified, tamper-evident cryptographic container that binds four independent data layers into a single root hash at the precise microsecond of a physical match outcome. The four layers are: (1) event extraction metadata; (2) raw video file evidence; (3) AI model weight state; and (4) active versioned ruleset epoch. The simultaneous binding creates a deterministic, mathematical barrier against past-posting fraud — any wagering timestamp after the root hash generation timestamp is automatically flagged as fraudulent. The system produces SCITT-compliant receipts per IETF RFC 9943 (June 2026) for regulatory submission to state gaming commissions.

### II.3 Detailed Technical Description

#### II.3.1 Four-Layer Binding Architecture

The SettlementProof is constructed by the following procedure executed atomically at the moment of match finish detection:

```
import hashlib
import time
from dataclasses import dataclass
```

```
from typing import Optional

@dataclass
class SettlementProofEnvelope:
    # Layer 1: Event Extraction
    event_id: str
    athlete_id_winner: str
    finish_type: str              # e.g., "rnc_choke" → maps to KILL
    t_sec: float                  # Match elapsed time at finish
    event_metadata_hash: str      # SHA-256(event_id + athlete_id + finish_type + t_s

    # Layer 2: Video Evidence
    video_file_path: str
    frame_start: int              # First frame of submission sequence
    frame_end: int                # Last frame (tap or referee stop)
    video_hash: str               # SHA-256(raw video file bytes, frame_start:frame_e

    # Layer 3: Model Integrity
    model_version: str            # e.g., "PGF_Scout_v4.2"
    model_weights_hash: str       # SHA-256(model weight file bytes at inference time

    # Layer 4: Regulatory Context
    rubric_epoch_seq: int         # Active epoch sequence number
    rubric_digest: str            # SHA-256 of active rubric (from RubricEpoch record

    # Root Hash — computed simultaneously across all four layers
    root_hash: str                # SHA-256(event_metadata_hash + video_hash + model_
    timestamp_utc: float          # Unix epoch microseconds — binding timestamp

    # Asymmetric Signature
    signature: Optional[str]      # ECDSA-P256 signature of root_hash by ContextOS pr
    public_key_fingerprint: str   # ContextOS ledger public key fingerprint for verif


def construct_settlement_proof(
    telemetry: GrapplingTelemetry,
    video_path: str,
    model_manifest: dict,
    active_epoch: 'RubricEpoch'
) -> SettlementProofEnvelope:
    """
    Atomic construction — all four hashes computed before any is committed.
    Timestamp is set AFTER all four layer hashes are computed.
    """
    h = hashlib.sha256

    # Layer 1
    event_str = f"{telemetry.event_id}{telemetry.athlete_id}{telemetry.transition_nam
    event_hash = h(event_str.encode()).hexdigest()

    # Layer 2
    with open(video_path, 'rb') as f:
        video_bytes = f.read()
    video_hash = h(video_bytes).hexdigest()

    # Layer 3
```

```
    model_hash = h(model_manifest['weights_path'].encode()).hexdigest()

    # Layer 4 — already stored in RubricEpoch
    rubric_digest = active_epoch.rubric_digest

    # Root Hash — simultaneous concatenation
    root_input = event_hash + video_hash + model_hash + rubric_digest
    root_hash = h(root_input.encode()).hexdigest()

    # Timestamp AFTER root hash — anti-past-posting guarantee
    timestamp = time.time()

    return SettlementProofEnvelope(
        event_id=telemetry.event_id,
        athlete_id_winner=telemetry.athlete_id,
        finish_type=telemetry.transition_name,
        t_sec=telemetry.t_sec,
        event_metadata_hash=event_hash,
        video_file_path=video_path,
        frame_start=0,  # populated by video indexer
        frame_end=0,
        video_hash=video_hash,
        model_version=model_manifest['version'],
        model_weights_hash=model_hash,
        rubric_epoch_seq=active_epoch.epoch_seq,
        rubric_digest=rubric_digest,
        root_hash=root_hash,
        timestamp_utc=timestamp,
        signature=None,  # populated by ContextOS signing service
        public_key_fingerprint=model_manifest['public_key_fp']
    )
```

### II.3.2 Past-Posting Elimination Mechanism

Past-posting fraud occurs when a wager is placed in the temporal window between a physical submission and its digital ledger registration. The SettlementProof eliminates this window as follows:

1. AI vision agent detects finish (`p_finish ≥ 0.85` threshold for auto-seal; `≥ 0.95` for autonomous settlement)
2. `construct_settlement_proof()` executes atomically — all four layer hashes computed in the same process frame
3. `timestamp_utc` is recorded **after** root_hash is computed — root_hash cannot be backdated
4. Root hash is immediately broadcast to sportsbook endpoints over authenticated WebSocket
5. Any wagering system that presents a bet with `wager_timestamp > proof_timestamp_utc` is rejected with `PAST_POSTING_DETECTED` exception
6. Any wagering system that presents a bet with `wager_timestamp < proof_timestamp_utc` but `bet_type` that resolves on the sealed finish is accepted as pre-finish (valid)

### II.3.3 Cryptographic Chaining of Rubric Epochs

The `RubricEpoch` chain provides the **fourth layer** of the SettlementProof and independently constitutes a sealed, tamper-evident ruleset history:

```
@dataclass
class RubricEpoch:
    epoch_seq: int               # Unique, auto-increment, MongoDB unique index enfo
    timestamp: float             # UTC creation timestamp
    prev_rubric_digest: str      # SHA-256 of the previous epoch's full document
    rubric_digest: str           # SHA-256(prev_rubric_digest + ruleset_json)
    ruleset_json: dict           # Complete point table, bonuses, penalties
    activated_by: str            # Human administrator ID
    chain_valid: bool            # Computed on load — False breaks settlement
```

**Chain Integrity Rule:** Each epoch's `rubric_digest` is computed as `SHA-256(prev_rubric_digest + JSON.stringify(ruleset_json))`. Any attempt to modify a historical epoch's `ruleset_json` produces a different `rubric_digest`, which breaks the chain at that epoch. The system performs chain integrity validation before admitting any new SettlementProof, rejecting the proof with `CHAIN_INTEGRITY_FAILURE` if any epoch in history has a mismatched digest.

**Genesis Fallback:** If the MongoDB persistence layer is unavailable, the system defaults to an in-memory hardcoded "genesis" epoch (the first-ever PGF ruleset) to prevent system paralysis. This genesis epoch has a `prev_rubric_digest` of `"0" * 64` (all zeros).

**MongoDB Persistence Indexes:**

```
// Enforced compound indexes — Exhibit B provides full schema
db.rubric_epochs.createIndex({ "epoch_seq": 1 }, { unique: true })
db.rubric_epochs.createIndex({ "timestamp": -1 })
db.rubric_epochs.createIndex({ "epoch_seq": 1, "timestamp": -1 })  // Latest-query op
```

### II.3.4 Asymmetric Verification Workflow (Sportsbook Integration)

1. **Finish Detection:** PGF Scout AI (agent consensus ≥ 2/3) flags `SUBMISSION_DETECTED`
2. **Envelope Construction:** `construct_settlement_proof()` executes atomically
3. **Digital Signing:** ContextOS private key signs `root_hash` using ECDSA-P256
4. **Broadcast:** Signed `SettlementProofEnvelope` transmitted to:
    - Licensed sportsbook WebSocket endpoints
    - State gaming commission regulatory ledger
    - PGF public settlement dashboard
    - Constellation Network blockchain anchor (optional preferred embodiment)
5. **Institutional Verification:** Sportsbook uses ContextOS public key to verify ECDSA signature; on success, match outcome is "cleared" for payout

6. **SCITT Receipt Generation:** System generates a SCITT-format receipt (per IETF RFC 9943) containing the root_hash, timestamp, and public key fingerprint for independent regulatory audit

### II.3.5 Lazy Reconstruction for Regulatory Queries

For performance during deep audit queries, the system implements lazy reconstruction: the active scoring rubric is rebuilt from the epoch chain only upon a formal verification request, not on every match event. The `export_as_json()` method produces a complete, chain-verified export of all epochs and settlement envelopes.

## II.4 Provisional Independent Claims

**Claim II-1 (Four-Layer Settlement Proof):** A computer-implemented method for cryptographic sports wagering settlement comprising: computing a first hash of event extraction metadata and match-elapsed timestamp; computing a second hash of raw video file bytes covering the frame range of the scored event; computing a third hash of AI model weight file bytes active at the time of event scoring; computing a fourth hash of an active versioned ruleset epoch document; concatenating said four hashes; computing a root hash of said concatenation; recording a settlement timestamp after said root hash computation; wherein any wagering event with a timestamp exceeding said settlement timestamp is rejected as fraudulent.

**Claim II-2 (Chained Epoch Integrity):** A computer-implemented system for maintaining immutable sports ruleset history comprising: an append-only database collection storing ruleset epoch records; wherein each epoch record contains a recursive hash pointer computed as a cryptographic hash of the preceding epoch's hash concatenated with the current ruleset document; wherein modification of any historical epoch record produces a hash mismatch detectable by sequential chain traversal; wherein chain integrity validation is performed before admitting any settlement proof.

**Claim II-3 (SCITT Settlement Receipt):** A method for generating regulatory-compliant sports settlement receipts comprising: constructing a tamper-evident settlement envelope binding event metadata, video evidence, AI model state, and versioned ruleset in a single cryptographic digest; digitally signing said digest with an asymmetric private key; generating a SCITT-format receipt comprising said digest, signing timestamp, and corresponding public key fingerprint; transmitting said receipt to state gaming commission regulatory endpoints.

**Claim II-4 (Past-Posting Barrier):** A system for preventing temporal fraud in sports wagering comprising: a cryptographic hash computed atomically at the moment of AI-detected match finish; a settlement timestamp recorded after said hash computation; a wagering endpoint configured to reject any bet resolution request presenting a wager timestamp later than said settlement timestamp; wherein said atomic hash computation creates a mathematically deterministic temporal barrier between physical match outcome and wagering system ledger update.

## PART III — INVENTION TARGET 3

## NON-DESTRUCTIVE CORRECTION OVERLAY METHOD WITH CRYPTOGRAPHIC RUBRIC REGISTRIES ("CONTEXTOS HONESTY KERNEL")

### III.1 Technical Field

This invention relates to a non-destructive data adjudication system for AI-driven sports officiating that preserves complete forensic audit trails by applying human corrections as new timestamped ledger entries (overlays) rather than destructive overwrites, while simultaneously maintaining a cryptographically chained ruleset registry that prevents silent alterations to scoring logic.

### III.2 Summary of the Invention

The **ContextOS Honesty Kernel** comprises two interlocking systems: (1) the **Non-Destructive Adjudication Engine**, which resolves conflicts between AI telemetry classifications and human ground-truth determinations without deleting historical state; and (2) the **Sealed Belief Registry**, which cryptographically locks scoring rules at each epoch transition, preventing mid-event ruleset manipulation. Together, these systems satisfy state gaming commission non-repudiation requirements and provide the forensic auditability required for regulated wagering data feeds.

### III.3 Detailed Technical Description

#### III.3.1 Non-Destructive Adjudication Engine

**Trust-State Discrimination Protocol:**

When an AI vision classification conflicts with a human official's ground-truth determination, the system executes the following non-destructive workflow:

```
from enum import Enum
from dataclasses import dataclass
from typing import Optional

class TrustState(str, Enum):
    AI_REVIEWED = "ai_reviewed"
    HUMAN_CONFIRMED = "human_confirmed"
    HUMAN_OVERRIDE = "human_override"
    CONFLICT_FLAGGED = "conflict_flagged"

@dataclass
class MatchEvent:
    event_id: str
    match_id: str
    athlete_id: str
    ai_classification: str          # Original AI output
    ai_confidence: float
```

```
    trust_state: TrustState
    ground_truth_classification: Optional[str] = None  # Human override value
    override_timestamp: Optional[float] = None
    override_admin_id: Optional[str] = None
    suppressed_for_public: bool = False   # True when overridden — AI output hidden f
    correction_reason: Optional[str] = None


class NonDestructiveAdjudicationEngine:
    """
    INVARIANT: No MatchEvent record is ever deleted or modified in-place.
    All corrections are new MatchEvent records with trust_state=HUMAN_OVERRIDE.
    The original AI record remains in the database with suppressed_for_public=True.
    """

    def apply_correction(
        self,
        original_event: MatchEvent,
        ground_truth: str,
        admin_id: str,
        reason: str
    ) -> MatchEvent:
        # Step 1: Flag original record as suppressed (NOT deleted)
        original_event.suppressed_for_public = True
        original_event.trust_state = TrustState.CONFLICT_FLAGGED
        self.db.update(original_event)  # In-place flag only — data preserved

        # Step 2: Create new overlay record
        correction_event = MatchEvent(
            event_id=self._generate_uuid(),
            match_id=original_event.match_id,
            athlete_id=original_event.athlete_id,
            ai_classification=original_event.ai_classification,  # Preserved for audi
            ai_confidence=original_event.ai_confidence,
            trust_state=TrustState.HUMAN_OVERRIDE,
            ground_truth_classification=ground_truth,
            override_timestamp=time.time(),
            override_admin_id=admin_id,
            suppressed_for_public=False,  # This record is the public-facing truth
            correction_reason=reason
        )
        self.db.insert(correction_event)  # New record — original untouched
        return correction_event
```

**ContextOS Override Gate (Three-Check Sequence):**

1. **Check 1 — Conflict Detection:** System compares `ai_classification` to `ground_truth_classification`; if they differ by more than a configurable tolerance (e.g., different `SubmissionTier`), a `CONFLICT_DETECTED` event is raised
2. **Check 2 — Human Authority Verification:** `admin_id` must be in the `PGF_CERTIFIED_OFFICIALS` registry before correction is accepted
3. **Check 3 — Chain Integrity Confirmation:** Before finalizing the correction overlay, system verifies current `RubricEpoch` chain integrity; a broken chain prevents any new corrections

until chain is repaired

### III.3.2 Assert-No-Silent-Code-Drift Function

A critical integrity check that detects unauthorized modifications to scoring logic at system startup and before every settlement:

```
import hashlib
import json
from pathlib import Path

CANONICAL_SCORING_HASH = "e3b0c44298fc1c149afb"  # Stored offline, updated only by au

def assert_no_silent_code_drift(scoring_module_path: str, active_epoch: RubricEpoch)
    """
    Detects unauthorized modifications to scoring logic between deployments.
    Called at: (1) system startup, (2) before every SettlementProof construction,
               (3) before any rubric_epoch write operation.

    Raises ContextOSIntegrityError if scoring code has been modified outside
    of an authorized epoch transition.
    """
    with open(scoring_module_path, 'rb') as f:
        current_hash = hashlib.sha256(f.read()).hexdigest()

    # Validate against epoch's expected code hash
    if current_hash != active_epoch.expected_code_hash:
        raise ContextOSIntegrityError(
            f"SILENT_CODE_DRIFT_DETECTED: "
            f"scoring module hash {current_hash[:16]}... "
            f"does not match epoch {active_epoch.epoch_seq} "
            f"expected hash {active_epoch.expected_code_hash[:16]}..."
        )

    # Secondary check: validate lethality constant is intact
    from scoring_engine import KILL_WEIGHT, BREAK_WEIGHT
    if KILL_WEIGHT != BREAK_WEIGHT * 2.0:
        raise ContextOSIntegrityError("LETHALITY_CONSTANT_CORRUPTED")

    return True
```

### III.3.3 Rubric Registry — Sealed Belief Architecture

The `RubricOverlay` document schema captures the correction event as a structured overlay linked to the current epoch:

```
{
  "overlay_id": "uuid-v4",
  "created_at": 1751587200.123456,
  "epoch_seq": 42,
  "rubric_digest": "sha256-of-active-epoch",
  "event_id_corrected": "original-ai-event-uuid",
  "original_classification": "heel_hook_inside",
```

```
  "corrected_classification": "woj_lock",
  "correction_type": "HUMAN_OVERRIDE",
  "admin_id": "referee-id-007",
  "reason": "Ankle torque direction — toehold not heel hook",
  "public_facing": true,
  "ai_record_suppressed": true,
  "chain_hash": "sha256-of-(prev_overlay_hash + this_overlay_json)"
}
```

**Query Engine — Point-in-Time Reconstruction:**
The system serves "corrected state" data using point-in-time logic: for any query timestamp `T`, the system returns the most recent overlay record active at `T`, defaulting to the original AI record if no overlay exists. This allows sportsbooks to receive real-time corrected data while regulators can query the full raw AI telemetry layer independently.

### III.4 Provisional Independent Claims

**Claim III-1 (Non-Destructive Overlay Method):** A computer-implemented method for adjudicating disputes in AI-driven sports officiating comprising: detecting a discrepancy between an AI vision classification and a human official ground-truth determination; suppressing said AI classification record from public-facing data outputs without deleting said record from the database; creating a new timestamped ledger entry containing the human ground-truth determination; serving said new entry as the authoritative scoring output; wherein said original AI record is preserved in the database for regulatory audit access.

**Claim III-2 (Trust-State Discrimination Protocol):** A system for managing AI-human conflict states in automated sports officiating comprising: a trust-state field associated with each scoring event record, said field taking one of a predefined set of enumerated values including AI_REVIEWED, HUMAN_CONFIRMED, HUMAN_OVERRIDE, and CONFLICT_FLAGGED; a non-destructive correction workflow that transitions a record to CONFLICT_FLAGGED state and creates a new HUMAN_OVERRIDE record upon official adjudication; wherein no scoring record is deleted from the persistence layer.

**Claim III-3 (Silent Code Drift Detection):** A computer-implemented method for detecting unauthorized modifications to sports officiating scoring logic comprising: computing a cryptographic hash of scoring module source code at system initialization; comparing said hash against an expected hash value stored within the current active ruleset epoch record; raising a programmatic integrity exception if said hashes do not match; preventing settlement proof generation until hash match is confirmed.

**Claim III-4 (Sealed Belief Registry):** A computer-implemented system for maintaining immutable sports officiating rules comprising: an append-only database collection wherein each record represents a versioned scoring ruleset sealed as a cryptographic epoch; a chaining protocol wherein each epoch's hash incorporates the hash of the preceding epoch; a validation gate that verifies complete chain integrity before permitting any match scoring or settlement operation; wherein said validation gate raises a terminal exception upon detection of any epoch having a non-matching hash.

## PART IV — INVENTION TARGET 4

## MULTI-AGENT PARALLEL SPORTS VISION ORCHESTRATION AND SYNTHESIS PIPELINE ("S.C.O.U.T. PIPELINE")

### IV.1 Technical Field

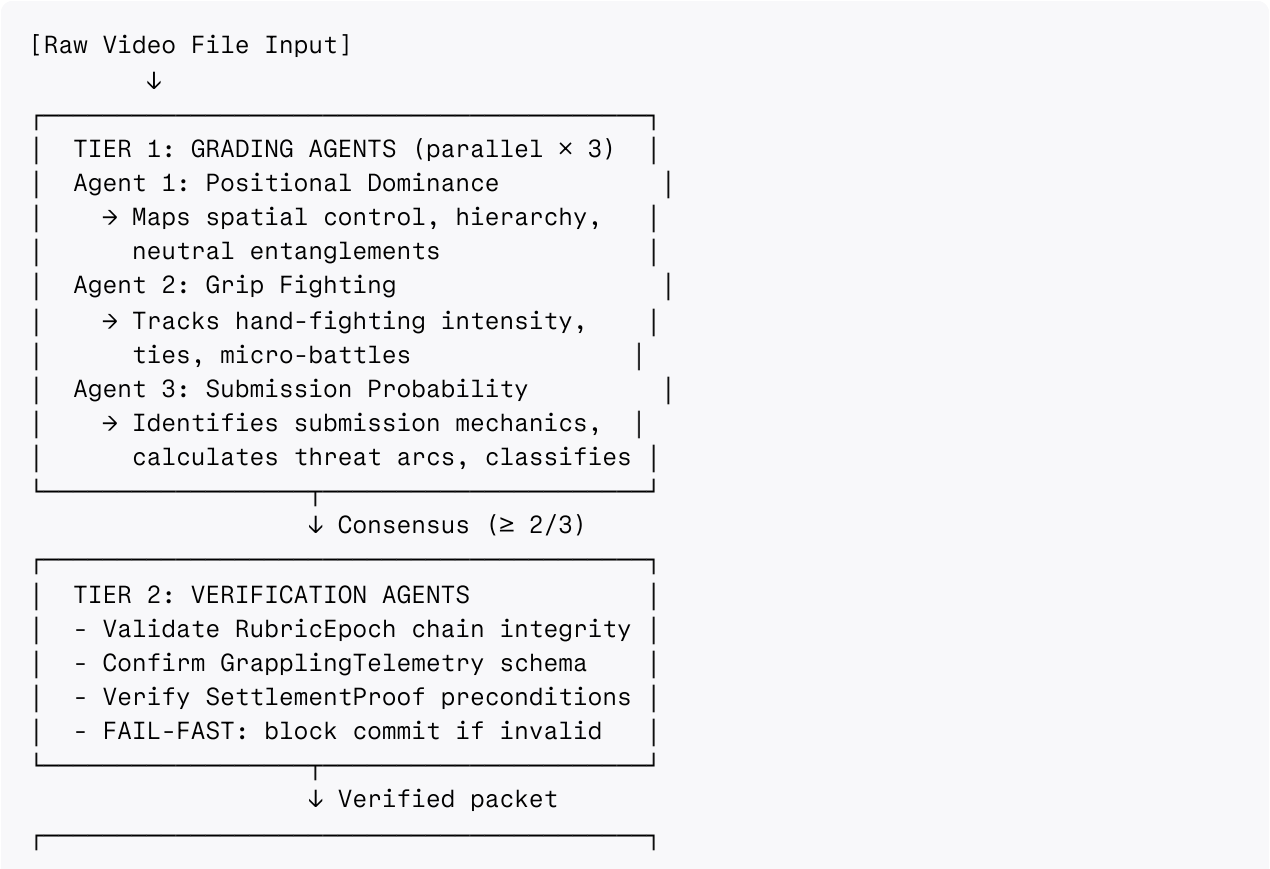
This invention relates to a distributed multi-agent artificial intelligence system for real-time, parallelized processing of combat sports video data, employing a three-tier agent hierarchy (Grading, Verification, Synthesis) to produce chain-verified, economically-settled outcome data from high-velocity, multi-surface competition environments.

### IV.2 Summary of the Invention

**S.C.O.U.T.** (Submission Combat Observation and Utility Tracking) is a multi-agent orchestration pipeline comprising three distinct agent classes operating in parallel: (1) Grading Agents performing binary submission classification; (2) Verification Agents performing fail-fast chain integrity validation before any leaderboard commit; and (3) Synthesis Agents aggregating verified outcomes into real-time XP leaderboards, team standings, and fantasy platform data feeds. The system is designed to process 30 matches within a 5.5-hour window across multiple simultaneous competition surfaces while maintaining data throughput and chain integrity.

### IV.3 Detailed Technical Description

#### IV.3.1 Three-Agent Pipeline Architecture



```
|  TIER 3: SYNTHESIS AGENTS                |
|  - XP Leaderboard update (real-time)     |
|  - Team Points calculation               |
|  - Fantasy PGF roster sync               |
|  - Franchise Dashboard broadcast         |
|  - Sportsbook settlement trigger         |
└──────────────────────────────────────────┘
Output: Pydantic-Validated Scouting Dossier
        in < 90 seconds per match
```

### IV.3.2 Kinematic Submission Probability Formula

The Grading Agent 3 computes `p_finish` using the following kinematic model:

$$P_{finish}(t) = \sigma\left(w_1 \cdot D_{pos}(t) + w_2 \cdot V_{trans}(t) + w_3 \cdot T_{arc}(t) + w_4 \cdot R_{escape}(t)^{-1} + b\right)$$

Where:

- $P_{finish}(t)$ = probability of submission finish at match time $t$
- $\sigma(\cdot)$ = logistic sigmoid activation function
- $D_{pos}(t)$ = positional dominance percentage at time $t$ (0.0–100.0)
- $V_{trans}(t)$ = transition velocity (positional changes per minute)
- $T_{arc}(t)$ = submission threat arc completion percentage (limb/neck mechanical angle)
- $R_{escape}(t)$ = escape rate coefficient derived from historical data for athlete archetype
- $w_1, w_2, w_3, w_4$ = learned weight coefficients (stored in active model weights, hashed in SettlementProof Layer 3)
- $b$ = bias term

**Settlement Thresholds:**

- `p_finish ≥ 0.85`: Auto-seal SettlementProof envelope (human review within 30 seconds)
- `p_finish ≥ 0.95`: Autonomous settlement (no human review required, direct sportsbook broadcast)
- `p_finish < 0.85`: Continue tracking, no settlement action

### IV.3.3 XP Engine and Fighter Card Generation

The XP Engine computes each athlete's total XP from their career match history against the active `RubricEpoch`:

$$XP_{total} = XP_{base} + \sum_{i=1}^{N} \left(S_i \cdot M_{tier}(i) + B_{elbow}(i)\right) + \Delta_{composite}$$

Where:

- $XP_{base} = 2000$ (baseline for all athletes entering the XP system)
- $S_i$ = score value of match event $i$ (3.0 for BREAK, 6.0 for KILL)

- $M_{tier}(i)$ = match tier multiplier (regular season = 1.0, playoff = 1.5, championship = 2.0)
- $B_{elbow}(i)$ = Elbow Genie bonus (1.0 if `t_sec < 60.0`, else 0.0)
- $\Delta_{composite}$ = composite adjustment from six diagnostic metrics

**Six-Metric Fighter Card Diagnostic:**

| Metric | Description | Range |
|---|---|---|
| `control_dominance` | Percentage of match time in dominant position | 0–100% |
| `transition_speed` | Average positional transitions per minute | 0–∞ |
| `sub_rate` | Submission attempts per 6-minute round equivalent | 0–∞ |
| `technique_diversity` | Unique submission types attempted per career | integer |
| `explosiveness` | Peak transition velocity measured in sub-5-second windows | 0–100% |
| `escape_rate` | Percentage of dominant positions successfully escaped by opponent | 0–100% |

**Archetype Classification:** Fighter Cards are classified into archetypes (e.g., "Blitz-Finisher", "Technical Scrambler", "Pressure Passer", "Defensive Specialist") based on the six-metric profile relative to the league-wide percentile distribution. The `Pro Analogue` field maps the athlete to the nearest matching historical PGF athlete profile (e.g., "Pro Analogue: Jett Thompson" for high sub_rate + low stall_risk profiles).

### IV.3.4 Exclusion Boundary and Agent Isolation

The three agent tiers operate with strict data exclusion boundaries to prevent tier contamination:

| Tier | Input Permitted | Output Permitted | Cross-Tier Access |
|---|---|---|---|
| Grading Agents | Raw video frames, calibration parameters | `GrapplingTelemetry` packets (pre-validation) | NONE |
| Verification Agents | `GrapplingTelemetry` packets, `RubricEpoch` chain | `VerifiedPacket` or `REJECTED` signals | Read-only: RubricEpoch chain |
| Synthesis Agents | `VerifiedPacket` objects only | XP updates, settlement triggers, broadcast events | Read-only: current standings |

No Grading Agent has read access to the `RubricEpoch` chain (prevents bias toward current point values). No Synthesis Agent has read access to raw video (prevents retrospective score manipulation).

## IV.4 Provisional Independent Claims

**Claim IV-1 (Multi-Agent Parallel Processing):** A system-implemented method for multi-agent parallel processing of combat sports action comprising: a plurality of Grading Agents configured to analyze raw video in parallel and classify submission events as either choke-based or joint-lock-based; a plurality of Verification Agents configured to perform fail-fast chain integrity validation of a versioned ruleset persistence layer before admitting any

classified event to a scoring ledger; a plurality of Synthesis Agents configured to aggregate verified events into a real-time athlete performance scoring system.

**Claim IV-2 (Fail-Fast Chain Verification Gate):** A computer-implemented method for ensuring sports data integrity comprising: receiving a validated telemetry packet from an interface layer; querying an append-only ruleset epoch collection to verify sequential chain integrity from genesis epoch to current epoch before committing said packet to any public-facing data store; blocking commitment and raising an integrity exception if any epoch in said chain has a hash mismatch.

**Claim IV-3 (Kinematic Finish Probability):** A computer-implemented method for computing real-time submission probability in combat sports comprising: applying a logistic sigmoid function to a linear combination of positional dominance percentage, transition velocity, submission arc completion percentage, and inverse escape rate coefficient; outputting a finish probability value between 0.0 and 1.0; triggering an autonomous settlement protocol when said probability value meets or exceeds a configurable threshold.

**Claim IV-4 (XP-Based Athlete Scoring):** A computer-implemented system for athletic performance quantification comprising: a baseline XP value assigned to each athlete upon entry into the system; an accumulative formula adding scored match events weighted by submission tier, match tier multiplier, and temporal bonus; a six-metric diagnostic profile comprising control dominance, transition speed, submission rate, technique diversity, explosiveness, and escape rate; an archetype classifier assigning each athlete to a named performance category based on percentile distribution of said six metrics across the league athlete population.

## PART V — INVENTION TARGET 5

## AUTOMATED ATHLETIC TALENT ACQUISITION AND MONETIZATION ENGINE ("DRAFT YOURSELF" + GLOBAL TALENT LEDGER)

### V.1 Technical Field

This invention relates to a direct-to-consumer automated athletic scouting platform that generates cryptographically-backed, standardized athlete research profiles from user-submitted combat sports video, employs a top-quintile filter to automatically issue professional tryout invitations, and creates a continuously self-reinforcing data moat through user-generated model training contributions.

### V.2 Summary of the Invention

The **Draft Yourself Engine** transforms the traditional combine model from a capital expenditure into a revenue-generating direct-to-consumer (DTC) product. Athletes pay access fees to submit match video to the S.C.O.U.T. pipeline, which generates a standardized AI Fighter Card within 90 seconds. Athletes scoring in the top 20th percentile (top quintile) of the `XP_combine` metric automatically receive professional tryout invitations. All user-submitted

video is retained as training data, creating a continuously expanding proprietary dataset that serves as a structural competitive moat.

### V.3 Detailed Technical Description

#### V.3.1 Five-Stage Draft Yourself Pipeline

```
Stage 1: SUBMISSION
  Fan/athlete uploads local gym match video to scout.pgf.world
  + pays access fee (DTC monetization event)
  + consents to training data license
        ↓
Stage 2: PROCESSING
  S.C.O.U.T. three-agent pipeline processes uploaded video
  GrapplingTelemetry packets generated, Pydantic-validated
  < 90 second target processing window
        ↓
Stage 3: FIGHTER CARD GENERATION
  Six-metric profile computed against XP_combine baseline
  Archetype classification assigned
  Pro Analogue matched from historical PGF athlete database
  XP_combine score calculated
        ↓
Stage 4: DISTRIBUTION
  Fighter Card published to athlete's profile page
  Athlete shares card on social media (zero-CAC viral marketing)
  Athlete automatically entered into Fantasy PGF as draftable character
  (Breaks 75-card professional roster ceiling with user-generated entries)
        ↓
Stage 5: TOP-QUINTILE FILTER + AUTO-INVITATION
  If XP_combine PERCENTILE_RANK(athlete) ≥ 80th percentile:
    → System automatically generates official PGF Tryout Invitation
    → Invitation delivered via email + app notification
    → Athlete profile flagged as "TRYOUT_INVITED" in Global Talent Ledger
```

#### V.3.2 Top-Quintile Automatic Trigger

```
def evaluate_draft_yourself_result(
    athlete_id: str,
    xp_combine: float,
    league_percentile_data: dict
) -> dict:
    """
    Automatic tryout invitation trigger — no human reviewer required for top-quintile
    """
    percentile_rank = compute_percentile_rank(xp_combine, league_percentile_data)

    result = {
        "athlete_id": athlete_id,
        "xp_combine": xp_combine,
        "percentile_rank": percentile_rank,
        "tryout_invited": False,
```

```
        "invitation_id": None
    }

    if percentile_rank >= 80.0:  # Top quintile threshold
        invitation = generate_tryout_invitation(athlete_id)
        result["tryout_invited"] = True
        result["invitation_id"] = invitation.id

        # Log to Global Talent Ledger (immutable append)
        talent_ledger.append({
            "event_type": "AUTO_TRYOUT_INVITATION",
            "athlete_id": athlete_id,
            "xp_combine": xp_combine,
            "percentile_rank": percentile_rank,
            "invitation_id": invitation.id,
            "timestamp": time.time(),
            "triggered_by": "AUTO_QUINTILE_FILTER"
        })

    return result
```

### V.3.3 Perpetual Data Moat Mechanics

The structural competitive advantage of the Draft Yourself system compounds over time:

1. **Training Data Accumulation:** Every user-submitted video (after consent) is added to the PGF Scout training corpus. The model becomes progressively more accurate at PGF-specific micro-events as corpus size grows.
2. **Proprietary Taxonomy Lock-In:** The 160+ member Closed Position Registry is calibrated specifically to PGF scoring logic. Models trained on this taxonomy cannot be directly substituted with generic BJJ vision tools.
3. **Historical Performance Ledger:** Every athlete who enters the Draft Yourself pipeline contributes to a historical XP baseline. As the ledger grows, the percentile normalization becomes more precise, making the top-quintile filter increasingly accurate.
4. **Viral Distribution Flywheel:** Athlete-shared Fighter Cards drive zero-cost user acquisition. New athletes attracted by shared cards pay access fees, submit videos, and contribute more training data — closing the flywheel loop.

### V.3.4 Regression-to-Mean Analysis and Fantasy Economic Integrity

The Synthesis Agent performs automated regression analysis to identify:

- **Overperforming Outliers** (athletes whose XP exceeds their archetype's historical mean by >1.5σ — potential breakout candidates)
- **Underperforming Premiums** (athletes whose XP falls below their archetype mean — potential salary cap inefficiencies)

This analysis feeds the Fantasy PGF economic model, ensuring that automated fantasy pricing reflects accurate performance probability rather than social media hype.

### V.4 Provisional Independent Claims

**Claim V-1 (Draft Yourself Pipeline):** A computer-implemented method for automated athletic talent evaluation comprising: receiving, from a user device, a combat sports match video file accompanied by a payment; processing said video through a multi-agent AI pipeline to generate a structured athlete performance profile comprising at least six diagnostic metrics; computing a percentile rank of the athlete's composite performance score relative to a database of previously evaluated athletes; automatically issuing a professional tryout invitation to athletes whose percentile rank meets or exceeds a threshold without requiring human reviewer intervention.

**Claim V-2 (Top-Quintile Auto-Invitation):** A computer-implemented system for automated sports talent identification comprising: a plurality of athlete performance profiles stored in a database, each profile containing a composite score derived from AI video analysis; a configurable percentile threshold defining a talent identification tier; an automated invitation generation module triggered upon any athlete profile achieving a percentile rank at or above said threshold; an immutable talent ledger recording each such automated invitation event.

**Claim V-3 (Viral Data Flywheel):** A computer-implemented system for combined athletic talent discovery and AI model improvement comprising: receiving user-submitted combat sports video files; generating shareable athlete profile cards from said video files; accepting user consent to retain said video files as machine learning training data; continuously adding consented video files to an AI model training corpus; wherein said corpus grows in direct proportion to platform user acquisition driven by shared athlete profile cards.

**Claim V-4 (Fantasy Economic Integration):** A computer-implemented method for automated sports fantasy market pricing comprising: generating athlete performance scores from AI video analysis; computing regression-to-mean deviation for each athlete relative to their classified archetype's historical score distribution; identifying athletes whose scores deviate by more than a threshold from archetype mean as outliers; adjusting fantasy market pricing for said athletes to reflect deviation-corrected performance probability rather than unadjusted historical scores.

## PART VI — PRIOR ART DISTINGUISHING TABLE

The following table formally distinguishes the present inventions from known prior art for prosecution record purposes:

| Prior Art Reference | Publication Number | What It Discloses | Key Distinction from Present Inventions |
|---|---|---|---|
| Intel Officiating System | US 7,005,970 B2 | Computer-aided sports officiating | Does not disclose Pydantic schema-constrained interface; no epoch chaining; no SettlementProof; no multi-agent consensus; no correction overlays |
| IBM Referee Gaze Patent | US 20170364753 A1 | Gaze-tracking referee assistance | Human-in-the-loop required; no closed position registry; no cryptographic settlement layer; no non-destructive adjudication |

| Prior Art Reference | Publication Number | What It Discloses | Key Distinction from Present Inventions |
|---|---|---|---|
| Simpson Officiating System | US 20200289887 A1 | Sensor boundary detection for sports | Physical sensor-based (not vision AI); no schema validation; no epoch persistence; no hallucination elimination |
| AI/ML Deterministic Module | US 20250342385 A1 | Non-deterministic AI with deterministic check module | Does not disclose sports-specific closed registry; no financial settlement application; no chained epoch persistence; no correction overlay |
| Goldman Sachs SETLcoin | US 20170287090 A1 | Blockchain-based financial settlement | Financial instruments only — no sports vision classification; no rubric epoch chaining; no multi-agent orchestration; no combat sports telemetry schema |

## PART VII — 35 U.S.C. § 112(a) ENABLEMENT STATEMENT

The specification herein, together with the exhibits incorporated by reference, provides sufficient description to enable a person of ordinary skill in the relevant arts (computer vision systems, distributed AI architectures, cryptographic ledger systems, sports data systems) to make and use the inventions claimed herein without undue experimentation. Specifically:

- The `GrapplingTelemetry` Pydantic v2 model is fully specified with all field types, validators, and constraints (Part I, §I.3.1; Exhibit A)
- The `SettlementProof` construction algorithm is fully specified with complete Python pseudocode (Part II, §II.3.1)
- The `RubricEpoch` chaining protocol is fully specified with hash computation formulas (Parts II and III)
- The `NonDestructiveAdjudicationEngine` workflow is fully specified with complete Python pseudocode (Part III, §III.3.1)
- The `assert_no_silent_code_drift()` function is fully specified with complete Python implementation (Part III, §III.3.2)
- The three-agent S.C.O.U.T. pipeline is fully specified with tier isolation boundaries (Part IV, §IV.3.1)
- The kinematic finish probability formula $P_{finish}(t)$ is fully specified with all variable definitions (Part IV, §IV.3.2)
- The XP engine formula is fully specified with all component terms defined (Part IV, §IV.3.3)
- The Draft Yourself five-stage pipeline is fully specified with all decision logic (Part V, §V.3.1–V.3.2)

The best mode known to the inventor for practicing the inventions as of the filing date of this provisional application is the system deployed as "PGF Scout" and "ContextOS" within the Professional Grappling Federation, operating on a stack comprising: Python 3.12 + Pydantic v2 (Interface Layer), MongoDB 7.0 (epoch persistence), Vercel serverless functions (API endpoints), React 18 (Athlete Research UI), and SHA-256 + ECDSA-P256 (cryptographic layer).

## PART VIII — ALICE/MAYO PATENT ELIGIBILITY DEFENSE (35 U.S.C. § 101)

Each claimed invention addresses a **specific technical problem** and provides a **specific technical improvement**, satisfying the *Alice Corp. v. CLS Bank Int'l* (2014) two-step framework and the August 2025 USPTO AI Eligibility Memorandum's requirement that AI inventions "improve another technology":

| Target | Technical Problem Solved | Technical Improvement Achieved |
|---|---|---|
| Target 1 (Salinas Architecture) | AI vision hallucinations corrupt financial settlement data | Eliminates hallucinations by hard-boundary schema rejection — not achievable by a human mind operating at match speed across 160 event types |
| Target 2 (SettlementProof) | Temporal fraud ("past-posting") in sports wagering | Creates a cryptographic temporal barrier via four-layer simultaneous SHA-256 binding — not a mathematical abstraction but a specific machine-implemented hash construction sequence |
| Target 3 (Honesty Kernel) | Silent data overwrites that void regulatory non-repudiation | Non-destructive overlay architecture with trust-state discrimination — a specific database operation sequence yielding forensic auditability unavailable in conventional destructive-write systems |
| Target 4 (S.C.O.U.T.) | Single-agent bottleneck and chain integrity failures in high-frequency sports data | Three-tier parallel multi-agent architecture with fail-fast chain validation gate — specific technical architecture improvement to distributed AI system throughput |
| Target 5 (Draft Yourself) | Manual talent scouting is subjective and unscalable for franchise operations | Automated top-quintile filter with viral data flywheel — specific system architecture creating a self-reinforcing proprietary training corpus not achievable by conventional scouting methods |

Per *Alice* Step 2A Prong Two (2024 AI Guidance), the additional elements in each claim — the Pydantic v2 validation library, SHA-256 hash construction, ECDSA-P256 signing, MongoDB append-only persistence, multi-agent consensus mechanism, kinematic sigmoid formula with specific combat sports variables — are not "well-understood, routine, conventional" when combined in this specific ordered arrangement for this specific technical application.

## PART IX — FILING CHECKLIST

The following exhibits must be attached before submission to USPTO Patent Center:

| Exhibit | Content | Status |
|---|---|---|
| **Exhibit A** | Complete `GrapplingTelemetry` Pydantic v2 model with all 160+ `PositionEnum` members | REQUIRED |
| **Exhibit B** | `RubricEpoch` and `RubricOverlay` MongoDB collection schemas with all compound index definitions | REQUIRED |
| **Exhibit C** | Sample `SettlementProofEnvelope` JSON document with all fields populated for a real PGF match event | REQUIRED |
| **Exhibit D** | Complete `assert_no_silent_code_drift()` implementation including `ContextOSIntegrityError` class | REQUIRED |

| Exhibit | Content | Status |
|---|---|---|
| **Exhibit E** | Mathematical derivation of $P_{finish}(t)$ kinematic formula with training methodology for weight coefficients | REQUIRED |
| **Exhibit F** | Complete XP engine formula derivation with sigmoid definition and baseline calibration methodology | REQUIRED |
| **Figure 1** | System architecture diagram: three-tier S.C.O.U.T. pipeline with data flow | REQUIRED (35 U.S.C. § 113) |
| **Figure 2** | SettlementProof four-layer binding diagram with hash construction flowchart | REQUIRED |
| **Figure 3** | Rubric epoch chaining protocol diagram with genesis fallback | REQUIRED |
| **Figure 4** | Non-destructive overlay flowchart with Trust-State Discrimination decision tree | REQUIRED |
| **Figure 5** | Draft Yourself five-stage pipeline diagram with viral flywheel loop | REQUIRED |

## PART X — 12-MONTH PROSECUTION ROADMAP

| Milestone | Date | Action Required |
|---|---|---|
| **Provisional Filed** | July 3, 2026 | Upload to USPTO Patent Center — this document + all exhibits |
| **Month 1–3** | Aug–Oct 2026 | File PCT application (WO jurisdiction) to preserve international rights |
| **Month 6** | Jan 2027 | Conduct formal prior art search; identify any additional distinguishing arguments |
| **Month 9** | April 2027 | Draft non-provisional applications — begin with Target 2 (SettlementProof, highest commercial value) and Target 3 (Honesty Kernel, regulatory moat) |
| **Month 11** | June 2027 | File first two non-provisional applications with complete claim sets (independent + 20–30 dependent per application) |
| **Month 12 — HARD DEADLINE** | **July 3, 2027** | All remaining non-provisional applications (Targets 1, 4, 5) must be filed before 11:59 PM EDT |

*Respectfully submitted,*

**Dustin Blaine Salinas**
Inventor
Las Vegas, Nevada
July 3, 2026

*This provisional application does not require a formal claim, oath, or declaration under 35 U.S.C. § 111(b). No prior art information disclosure statement is required or included.*

# PART VI-A - SPECIAL SYSTEM SPECIFICATION

# CRYPTOGRAPHICALLY SECURED REAL-TIME SPORTS WAGERING AND DETERMINISTIC IN-PLAY SETTLEMENT

## VI-A.1 Technical Field

This embodiment relates to real-time distributed data processing, automated event extraction, and cryptographic data validation. More specifically, this embodiment relates to a system and method for ingesting raw combat sports video feeds, executing a multi-agent computer vision pipeline to extract continuous athletic performance and state-transition telemetry, validating the extracted telemetry against a ruleset schema, and generating a cryptographically signed SettlementProof envelope to authorize or settle high-frequency in-play sports wagering markets without dependency on an unverified human oracle.

## VI-A.2 Real-Time In-Play Sportsbook Settlement Flow

Raw video frame feed is processed by S.C.O.U.T. parallel agents for pose, positional, grip, submission, and telemetry extraction. ContextOS adjudication validates the extracted data against the active GENESIS_RUBRIC_DIGEST and closed registry. A trust-state filter applies a non-destructive official override overlay when required. A SettlementProof builder binds frame hash, ruleset digest, model integrity, and score or market event payload. An ECDSA-P256 or equivalent asymmetric signature module signs the resulting payload for sportsbook API delivery.

## VI-A.3 Real-Time Telemetry and In-Play Micro-Markets

The system converts continuous physical combat activity into structured low-latency telemetry. A Submission Probability Agent tracks skeletal joint deflections, limb entanglement states, neck target capture states, control dominance states, transition velocity, and escape-rate indicators. These physical states are processed against the PGF Grapple Map taxonomy and the active rubric epoch. The telemetry supports deterministic in-play prop markets including active submission threat probability markets, positional momentum and control-dominance markets, and shot-clock or activity-window markets.

## VI-A.4 Cryptographic SettlementProof Envelope

At or near the time of an event eligible for market settlement, the ContextOS backend compiles a SettlementProof envelope. The envelope binds at least four data layers into one cryptographic payload: an event ingestion layer, an evidence pack layer including a SHA-256 hash of the raw video frame range or evidence clip containing the physical event, a model and rubric integrity layer, and a temporal layer. The envelope is signed with an asymmetric signing key and verified by a sportsbook, regulator, league auditor, or verification endpoint.

## VI-A.5 Silent Code-Drift Honesty Guard

Before writing a scoring entry, market tick, market settlement, or SettlementProof to a database or external feed, the backend may execute a code-drift guard. The guard recomputes a SHA-256 digest of the active scoring code and compares the digest to the registered genesis or active rubric digest. If the digests do not match, the system blocks the scoring or settlement operation, quarantines the match or market identifier, and records the failed verification state for auditor review.

## VI-A.6 Kinematic Finish Probability

In one embodiment, the AI-driven event extraction processor calculates a continuous kinematic finish probability P_finish(t) for active submission attempts by applying a logistic sigmoid function to a weighted combination of positional dominance percentage D_pos(t), transition velocity V_trans(t), submission target angular deflection T_arc(t), and inverse escape-rate coefficient R_escape(t)^-1.

```
P_finish(t) = sigma(w1 * D_pos(t) + w2 * V_trans(t) + w3 * T_arc(t) + w4 * R_escape(t)^-1 + b)
```

where w1, w2, w3, and w4 are trained coefficients and b is a bias value stored in an active AI model weight file or model configuration file.

## VI-A.7 Additional Claim Embodiments

9. A computer-implemented system for executing and cryptographically settling real-time, in-play sports wagers on high-frequency combat sports events, comprising a video capture apparatus, an AI-driven event extraction processor configured to run concurrent non-overlapping computer vision agents in parallel, a ruleset-native scoring engine, a context adjudication processor configured to execute a silent code-drift guard and apply a non-destructive referee data override overlay, and a cryptographic settlement engine configured to compile, bind, sign, and transmit a unified tamper-evident SettlementProof envelope to connected sportsbook client endpoints.

10. The system of claim 9, wherein the AI-driven event extraction processor calculates a continuous kinematic finish probability P_finish(t) for active submission attempts by applying a logistic sigmoid function to a linear combination of positional dominance percentage, transition velocity, submission target angular deflection, and inverse escape-rate coefficient.

11. The system of claim 10, wherein a finish probability value within a first threshold range triggers automatic SettlementProof envelope generation with a human review flag, and wherein a finish probability value above a second higher threshold triggers autonomous machine-settled wager resolution subject to run-mode policy and scoped settlement approval requirements.

12. The system of claim 9, wherein the context adjudication processor writes ruleset modifications as sequentially numbered, cryptographically linked Rubric Epochs, wherein each epoch is sequentially bound to a prior epoch cryptographic digest, allowing historical match records to be deterministically replayed and audited under the exact ruleset epoch active on their respective match dates.

13. The system of claim 12, wherein connected sportsbook client endpoints are configured to run live, high-frequency micro-betting prop markets comprising at least one of a probability-threshold submission market, a rolling positional-control dominance market, and an activity-window or shot-clock violation market, each settled or voided according to a cryptographic SettlementProof and active market policy.

# PROVISIONAL PATENT APPLICATION — PART 2

### Dependent Claims, Complete Exhibit Set, and Prosecution Support Documents

**Application Title:** DETERMINISTIC COMBAT SPORTS OFFICIATING SYSTEM WITH CRYPTOGRAPHIC SETTLEMENT PROOF, SCHEMA-CONSTRAINED VISION TELEMETRY, NON-DESTRUCTIVE CORRECTION OVERLAYS, MULTI-AGENT PARALLEL SYNTHESIS PIPELINE, AND AUTOMATED TALENT ACQUISITION ENGINE

**Inventor:** Dustin Blaine Salinas, Las Vegas, Nevada
**Priority Date:** July 3, 2026
**This document supplements the Part 1 specification filed simultaneously herewith.**

## SECTION A — COMPLETE DEPENDENT CLAIMS

> **Prosecution Strategy Note:** Dependent claims serve three functions in a patent portfolio: (1) they provide fallback positions if an independent claim is rejected during prosecution; (2) they are the primary tool for blocking design-arounds by competitors who engineer narrowly around independent claims; and (3) in litigation, they allow damages to be apportioned to specific infringing features. File all dependent claims listed below. Do not cut them for brevity.

## TARGET 1 DEPENDENT CLAIMS

### (Dependent on Independent Claims I-1 through I-5)

**Claim I-6** (dependent on I-1): The method of claim I-1, wherein the closed registry comprises exactly the following taxonomic groups: CHOKE_FINISH_LABELS, JOINT_LOCK_FINISH_LABELS, DOMINANT_POSITIONS, NEUTRAL_ENTANGLEMENTS, TEMPO_EVENT_LABELS, and GripTypeLiteral.

**Claim I-7** (dependent on I-1): The method of claim I-1, wherein the constrained interface layer is implemented using the Pydantic v2 Python library and wherein validation failure raises a `ValidationError` exception that prevents any downstream scoring operation.

**Claim I-8** (dependent on I-1): The method of claim I-1, wherein the scoring engine applies point weights from a file stored outside of neural network weight files, wherein said file is identified by an active epoch sequence number in an append-only persistence layer.

**Claim I-9** (dependent on I-2): The method of claim I-2, wherein the predefined closed registry comprises at minimum: (a) twelve (12) choke submission labels including `rear_naked_choke`, `guillotine`, `triangle`, `arm_triangle`, `north_south_choke`, `bow_and_arrow`, `peruvian_necktie`, `japanese_necktie`, and `ezekiel`; (b) fifteen (15) joint lock submission labels including `armbar`,

`kimura`, `americana`, `heel_hook_inside`, `heel_hook_outside`, `straight_ankle_lock`, `kneebar`, `toe_hold`, `estima_lock`, `wristlock`, `omoplata`, `calf_slicer`, `electric_chair`, `banana_split`, and `twister_finish`; (c) ten (10) dominant position labels including `mount`, `back_control`, `side_control`, `knee_on_belly`, `saddle_411`, and `inside_sankaku`; and (d) six (6) neutral entanglement labels including `outside_ashi`, `crab_ride`, and `50_50_guard`.

**Claim I-10** (dependent on I-2): The method of claim I-2, wherein the predefined closed registry additionally comprises eight (8) temporal event labels including `ref_position_reset`, `shot_clock_warning`, and `avoidance_penalty`, stored at the backend path `backend/scout/score/metrics.py`.

**Claim I-11** (dependent on I-2): The method of claim I-2, wherein the predefined closed registry additionally comprises eighteen (18) grip type labels including `collar_tie`, `sleeve_grip`, `wrist_control`, `underhook`, and `overhook`.

**Claim I-12** (dependent on I-3): The scoring engine of claim I-3, wherein any vision agent output classified as `twister_finish` is enforced as a JOINT_LOCK_FINISH_LABELS member and is systematically blocked from assignment to the CHOKE_FINISH_LABELS category by the constrained interface layer regardless of vision agent confidence score.

**Claim I-13** (dependent on I-3): The scoring engine of claim I-3, further comprising a temporal bonus module that awards one (1) additional point to any submission finishing event wherein the match elapsed time at detection is less than sixty (60) seconds, wherein said elapsed time is validated against the `t_sec` field of the telemetry schema.

**Claim I-14** (dependent on I-3): The scoring engine of claim I-3, wherein the lethality constant of 2.0 is enforced programmatically via an assertion statement executed at engine initialization and re-executed before every scoring operation, and wherein any modification to said constant triggers a system-wide lockdown of all downstream database write operations.

**Claim I-15** (dependent on I-4): The functional component of claim I-4, wherein the D'Arce choke submission is represented exclusively as one of the string literals `d_arce_finish` or `d_arce_attempt`, and wherein any alternative representation including `darce_finish`, `darce_attempt`, or representations containing an apostrophe character are rejected by the registry validation function.

**Claim I-16** (dependent on I-4): The functional component of claim I-4, wherein the temporal event identifier `sub_under_60s` is excluded from TEMPO_EVENT_LABELS and is instead enforced as a native integer field `sub_under_60s_count` on the root validation model, wherein said field accepts only integer values of zero (0) or one (1).

**Claim I-17** (dependent on I-4): The functional component of claim I-4, wherein any vision agent output presenting a confidence score below a configurable threshold is routed to a human-in-the-loop override queue designated as `low_confidence_field`, and wherein said output is not processed by the downstream scoring engine until human confirmation is received.

**Claim I-18** (dependent on I-5): The method of claim I-5, wherein the append-only persistence layer is implemented as a MongoDB collection with a unique compound index enforcing uniqueness on the `epoch_seq` field.

**Claim I-19** (dependent on I-5): The method of claim I-5, wherein said scoring weights include point coefficients for at minimum one seasonal ruleset variation, and wherein point values from any previous epoch are independently accessible by specifying that epoch's sequence number without modification to the current active epoch.

**Claim I-20** (dependent on I-5): The method of claim I-5, further comprising a visual inference deployment system (Gemini 3.1 Pro Vision or equivalent) as the underlying AI vision model, wherein said model outputs are subject to the schema validation of claim I-1 and wherein the vision model's weight files are cryptographically hashed to produce a model integrity layer for inclusion in a downstream settlement proof.

## TARGET 2 DEPENDENT CLAIMS

### (Dependent on Independent Claims II-1 through II-4)

**Claim II-5** (dependent on II-1): The method of claim II-1, wherein the four hash values are computed using SHA-256 cryptographic hash function applied to: (a) a UTF-8 encoded concatenation of event identification string, athlete identification string, finish type string, and match elapsed time decimal; (b) raw binary bytes of a video file segment spanning a frame range identified by the vision pipeline; (c) raw binary bytes of AI model weight files active at inference time; and (d) a UTF-8 encoded JSON serialization of the active ruleset epoch document.

**Claim II-6** (dependent on II-1): The method of claim II-1, wherein the root hash is computed as SHA-256 of the UTF-8 encoded concatenation of all four individual layer hash hexdigest strings, and wherein the settlement timestamp is assigned from a system clock call executed in the same process frame as the root hash computation.

**Claim II-7** (dependent on II-1): The method of claim II-1, further comprising digital signing of the root hash using ECDSA with curve P-256, wherein the signing private key is held exclusively by the ContextOS ledger service and wherein the corresponding public key is published to a regulatory endpoint accessible to state gaming commissions.

**Claim II-8** (dependent on II-1): The method of claim II-1, wherein the settlement proof is broadcast in real time to one or more of: a licensed sportsbook WebSocket endpoint, a state gaming commission regulatory ledger, a public-facing league settlement dashboard, and a blockchain anchor service.

**Claim II-9** (dependent on II-2): The system of claim II-2, wherein each epoch record contains: (a) an integer sequence number enforced as unique by a database index; (b) a creation timestamp in Unix epoch format; (c) a hash pointer computed as SHA-256 of the concatenation of the preceding epoch's hash and the current ruleset JSON; and (d) an administrator identification field recording the human actor authorizing the epoch creation.

**Claim II-10** (dependent on II-2): The system of claim II-2, wherein the genesis epoch is identified by a preceding epoch hash field containing a string of sixty-four (64) zero characters

representing the initial chain anchor, and wherein the genesis epoch ruleset represents the foundational scoring rules from which all subsequent epochs derive.

**Claim II-11** (dependent on II-2): The system of claim II-2, further comprising a genesis fallback mechanism wherein, upon database unavailability, the system loads the genesis epoch from an in-memory hardcoded structure to prevent system paralysis, and wherein all settlement proofs generated under the fallback mechanism are flagged with a `FALLBACK_EPOCH` indicator for regulatory review.

**Claim II-12** (dependent on II-3): The method of claim II-3, wherein the SCITT-format receipt conforms to the IETF draft-ietf-scitt-architecture specification and contains at minimum: the settlement root hash, the UTC settlement timestamp, the ContextOS public key fingerprint, and a sequence identifier linking the receipt to the active rubric epoch.

**Claim II-13** (dependent on II-3): The method of claim II-3, further comprising transmission of the SCITT receipt to a blockchain anchoring service, wherein the receipt hash is committed to a distributed ledger block, and wherein the resulting block hash and block height are appended to the settlement proof record as additional audit fields.

**Claim II-14** (dependent on II-4): The system of claim II-4, wherein the autonomous settlement threshold is a configurable floating-point value between 0.0 and 1.0, wherein the default autonomous settlement threshold is 0.95 and wherein any finish probability value between 0.85 and 0.95 triggers a settlement proof generation with a human review flag set to true, allowing a certified official to confirm or override within a configurable review window.

**Claim II-15** (dependent on II-4): The system of claim II-4, further comprising lazy reconstruction logic wherein the complete epoch chain is rebuilt from genesis to current only upon receipt of a formal regulatory audit request, and wherein normal settlement operations use a cached current-epoch hash rather than traversing the full chain.

**Claim II-16** (dependent on II-4): The system of claim II-4, further comprising a `SettlementProof.export_as_json()` method that produces a complete, chain-verified export of all epochs and settlement envelopes associated with a specified match identifier, wherein said export is formatted for submission to state gaming commission audit portals.

## TARGET 3 DEPENDENT CLAIMS

### (Dependent on Independent Claims III-1 through III-4)

**Claim III-5** (dependent on III-1): The method of claim III-1, wherein the suppression operation sets a boolean field `suppressed_for_public` to `True` on the original AI classification record without modifying any other field of said record.

**Claim III-6** (dependent on III-1): The method of claim III-1, wherein the new timestamped ledger entry retains the original AI classification value in a dedicated field alongside the human ground-truth value, enabling machine learning model retraining systems to analyze the delta between AI output and verified ground truth for model improvement purposes.

**Claim III-7** (dependent on III-1): The method of claim III-1, further comprising a point-in-time query interface wherein, for any specified query timestamp T, the system returns the most recent overlay record active at T, defaulting to the original AI record if no overlay record exists for that time period.

**Claim III-8** (dependent on III-2): The system of claim III-2, wherein the CONFLICT_FLAGGED trust state is set on the original AI record simultaneously with the creation of the HUMAN_OVERRIDE record, and wherein said state transition is executed as an atomic database transaction to prevent any intermediate state where neither the original nor the correction is authoritative.

**Claim III-9** (dependent on III-2): The system of claim III-2, wherein the HUMAN_OVERRIDE trust state can only be written by an administrator identity present in a PGF_CERTIFIED_OFFICIALS registry, and wherein any attempt to write a HUMAN_OVERRIDE record by an unregistered identity raises an authorization exception and is logged to an immutable audit trail.

**Claim III-10** (dependent on III-2): The system of claim III-2, wherein a Heartbeat Consent Token — an externally signed security token with a configurable expiration window — must accompany every database write operation, and wherein detection of an expired or invalid token immediately revokes all pending write authorizations and triggers a system alert.

**Claim III-11** (dependent on III-3): The method of claim III-3, wherein the cryptographic hash of scoring module source code is computed as SHA-256 of the raw binary bytes of the scoring module file at the path `backend/scout/score/metrics.py`, and wherein said hash is stored in the `expected_code_hash` field of the active RubricEpoch record at epoch creation time.

**Claim III-12** (dependent on III-3): The method of claim III-3, further comprising a secondary integrity check that validates the lethality constant invariant by programmatically asserting that the KILL scoring weight is exactly 2.0 times the BREAK scoring weight, and wherein failure of said assertion triggers the same system-wide lockdown as a hash mismatch.

**Claim III-13** (dependent on III-3): The method of claim III-3, wherein the `assert_no_silent_code_drift()` function is invoked at: (a) system initialization; (b) before every SettlementProof construction; and (c) before any rubric epoch write operation; and wherein any invocation producing a mismatch writes a `CRITICAL: SILENT_CODE_DRIFT_DETECTED` event to an append-only security audit log before triggering system lockdown.

**Claim III-14** (dependent on III-4): The system of claim III-4, wherein the chaining protocol computes each epoch's hash as SHA-256 of the UTF-8 encoded concatenation of the preceding epoch's hash string and the JSON serialization of the current epoch's complete ruleset document.

**Claim III-15** (dependent on III-4): The system of claim III-4, wherein a `ContextOS Override Gate` enforces three sequential checks before any correction is finalized: (a) conflict detection comparing AI classification to human ground truth; (b) authority verification of the human reviewer's identity against a certified officials registry; and (c) chain integrity confirmation verifying that no epoch in the historical chain has a hash mismatch.

**Claim III-16** (dependent on III-4): The system of claim III-4, further comprising a `RubricOverlay` document schema stored in the database alongside the `RubricEpoch` chain, wherein each overlay document contains: the overlay UUID, the epoch sequence number active at correction time, the rubric digest of said epoch, the event identifier of the corrected record, the original AI classification, the corrected human classification, the correction type, the authorizing administrator identifier, the correction timestamp, and a chain hash linking overlays in chronological sequence.

**Claim III-17** (dependent on III-4): The system of claim III-4, further comprising a `Lie-Before-Action Check` that compares an AI agent's proposed output against immutable source records before any write operation, wherein detection of a contradiction, omission, or unsupported assertion causes the proposed action to be blocked and the agent's session token to be revoked.

## TARGET 4 DEPENDENT CLAIMS

### (Dependent on Independent Claims IV-1 through IV-4)

**Claim IV-5** (dependent on IV-1): The system of claim IV-1, wherein the Grading Agents comprise: (a) a PositionalDominanceAgent that owns the position timeline, dominant athlete identification, control_dominance scores, escape rates, and positional tempo events, and is explicitly barred from detecting submissions, grip exchanges, or athlete archetypes; (b) a GripFightingAgent that owns standing and ground grip exchanges, grip break velocity, engagement_intensity scores, technique diversity inputs, and avoidance penalties, and is explicitly barred from detecting submissions, positional control time, or match-ending finishes; and (c) a SubmissionProbabilityAgent that owns technique classification of twelve (12) choke types and fourteen (14) joint lock types, attempt versus finish status, time-to-finish, and sub-60s counts, and is explicitly barred from tracking positional control time or independent grip exchanges.

**Claim IV-6** (dependent on IV-1): The system of claim IV-1, wherein the multi-agent ecosystem is powered by a multimodal vision model, wherein each of the three Grading Agents runs as an isolated instance of said model with a dedicated system prompt restricting its domain of detection, and wherein cross-agent communication occurs only through the consensus mechanism at the Verification Agent tier.

**Claim IV-7** (dependent on IV-1): The system of claim IV-1, wherein the complete pipeline from raw video input to Pydantic-validated scouting dossier output operates within a target processing window of ninety (90) seconds per match, and wherein the system is architected to process at minimum thirty (30) matches within a five-and-one-half (5.5) hour event window across multiple concurrent competition surfaces.

**Claim IV-8** (dependent on IV-2): The method of claim IV-2, wherein the database compound index enforced before every commit operation is structured as `{"epoch_seq": 1, "timestamp": -1}` and wherein the uniqueness constraint on said index prevents duplicate event records from concurrent agent submissions.

**Claim IV-9** (dependent on IV-3): The method of claim IV-3, wherein the kinematic finish probability formula is:

P_finish(t) = sigmoid(w1 * D_pos(t) + w2 * V_trans(t) + w3 * T_arc(t) + w4 * R_escape(t)^(-1) + b)

wherein D_pos(t) is the positional dominance percentage, V_trans(t) is the transition velocity in positional changes per minute, T_arc(t) is the submission threat arc completion percentage measured as the angular progression of a limb or neck toward a mechanical finish position, R_escape(t) is the escape rate coefficient, and w1 through w4 and b are trained coefficients stored in the active AI model weight file.

**Claim IV-10** (dependent on IV-3): The method of claim IV-3, wherein a finish probability value between 0.85 and 0.95 exclusive triggers an automatic settlement proof envelope generation with a human review flag, and wherein a finish probability value of 0.95 or greater triggers autonomous settlement without requiring human review.

**Claim IV-11** (dependent on IV-4): The system of claim IV-4, wherein the baseline XP value of two thousand (2000) is assigned to every athlete upon their first appearance in the system, and wherein the post-match XP adjustment is calculated as a function of the synthesized match performance points earned under the asymmetric ruleset multiplied by a match tier multiplier.

**Claim IV-12** (dependent on IV-4): The system of claim IV-4, wherein the six diagnostic metrics comprising the Fighter Card are: control_dominance expressed as a percentage of match time in dominant position; transition_speed expressed as average positional transitions per minute; sub_rate expressed as submission attempts per six-minute round equivalent; technique_diversity expressed as the count of unique submission types attempted over the athlete's career history; explosiveness expressed as peak transition velocity measured in sub-five-second windows; and escape_rate expressed as the percentage of dominant positions successfully escaped by the opponent.

**Claim IV-13** (dependent on IV-4): The system of claim IV-4, wherein the archetype classification assigns each athlete to one of a predefined set of named archetypes based on the athlete's percentile distribution across the six diagnostic metrics relative to the full league athlete population, and wherein the Fighter Card additionally includes a Pro Analogue field identifying the nearest matching historical PGF athlete profile by Euclidean distance in the six-metric feature space.

**Claim IV-14** (dependent on IV-4): The system of claim IV-4, wherein the XP leaderboard, team standings, and fantasy platform data feeds are updated synchronously upon completion of each Synthesis Agent processing cycle, and wherein said updates are pushed to connected client endpoints via WebSocket without requiring client polling.

## TARGET 5 DEPENDENT CLAIMS

### (Dependent on Independent Claims V-1 through V-4)

**Claim V-5** (dependent on V-1): The method of claim V-1, wherein the structured athlete performance profile additionally includes an archetype classification, a Pro Analogue match, and an XP_combine score representing the athlete's composite performance relative to a normalization baseline derived from the database of previously evaluated athletes.

**Claim V-6** (dependent on V-1): The method of claim V-1, wherein the payment is processed as a direct-to-consumer access fee at the portal domain scout.pgf.world, and wherein the payment receipt is associated with the athlete's submitted video file as a prerequisite for pipeline processing.

**Claim V-7** (dependent on V-1): The method of claim V-1, further comprising a standardized physical combine module that measures at minimum: isometric plank endurance, dead-hang grip endurance, and multi-directional cone shuttle agility, and wherein said measurements are appended to the athlete's profile as biometric data fields supplementing the video-derived six-metric diagnostic.

**Claim V-8** (dependent on V-2): The system of claim V-2, wherein the configurable percentile threshold defaulting to the eightieth (80th) percentile defines the top quintile boundary, and wherein the automated invitation generation module produces an invitation document containing: the athlete's name, their XP_combine score, their percentile rank, the date of the next scheduled PGF sanctioned tryout event, and a unique invitation identifier stored in the Global Talent Ledger.

**Claim V-9** (dependent on V-2): The system of claim V-2, wherein the Global Talent Ledger is implemented as an append-only database collection wherein each invitation event record is immutable after creation and wherein the complete invitation history for any athlete is reconstructable from the append-only record sequence.

**Claim V-10** (dependent on V-2): The system of claim V-2, further comprising a strategic lock-out mechanism wherein PGF maintains the exclusive database of AI-scored, unsigned amateur talent, and wherein no competing organization has access to the proprietary XP_combine scores or Fighter Card profiles of athletes evaluated through the Draft Yourself pipeline.

**Claim V-11** (dependent on V-3): The system of claim V-3, wherein user consent to training data retention is obtained through an explicit terms-of-service acceptance at the time of payment and video upload, and wherein said consent is recorded in a separate consent audit log linked to the athlete's submission record.

**Claim V-12** (dependent on V-3): The system of claim V-3, further comprising a fantasy sports integration module wherein every athlete with a published Fighter Card is automatically registered as a draftable character in the Fantasy PGF league, wherein said integration expands the draftable roster beyond the fixed professional athlete roster ceiling, and wherein fantasy users may draft amateur athletes based on their AI-generated performance metrics.

**Claim V-13** (dependent on V-4): The method of claim V-4, further comprising a regression-to-mean analysis module that identifies overperforming outliers as athletes whose XP_combine score exceeds their archetype's historical mean by more than 1.5 standard deviations, and

underperforming premiums as athletes whose score falls more than 1.5 standard deviations below their archetype mean.

**Claim V-14** (dependent on V-4): The method of claim V-4, wherein the fantasy market pricing adjustment is applied prospectively from the date of outlier identification and does not retroactively alter past fantasy scoring, consistent with the non-destructive overlay principle.

### SECTION B — EXHIBIT A: COMPLETE GrapplingTelemetry PYDANTIC v2 MODEL

> **Filing instruction:** Copy this code block verbatim into a file named `Exhibit_A_GrapplingTelemetry_Model.py` and attach to USPTO Patent Center submission as a computer program listing per 37 C.F.R. § 1.96.

```
# Exhibit A — GrapplingTelemetry Pydantic v2 Schema
# PGF Scout ContextOS — Canonical Model as of Filing Date July 3, 2026
# Inventor: Dustin Blaine Salinas
# CONFIDENTIAL — PATENT EXHIBIT

from pydantic import BaseModel, Field, field_validator, model_validator
from typing import Literal, Dict, Any, Optional
from enum import Enum
import time

# ─────────────────────────────────────────────────────────────────
# LABEL GROUPS (source: backend/scout/vision/gemini31_agents.py)
# ─────────────────────────────────────────────────────────────────

CHOKE_FINISH_LABELS = frozenset([
    "rear_naked_choke",
    "guillotine",
    "triangle",
    "arm_triangle",
    "north_south_choke",
    "bow_and_arrow",
    "peruvian_necktie",
    "japanese_necktie",
    "ezekiel",
    "d_arce_finish",        # ENFORCED: apostrophe variant rejected
    "anaconda_choke",
    "loop_choke",
])  # 12 members

JOINT_LOCK_FINISH_LABELS = frozenset([
    "armbar",
    "kimura",
    "americana",
    "heel_hook_inside",
    "heel_hook_outside",
    "straight_ankle_lock",
    "kneebar",
    "toe_hold",
    "estima_lock",
    "wristlock",
```

```
    "omoplata",
    "calf_slicer",
    "electric_chair",
    "banana_split",
    "twister_finish",       # ENFORCED: classified as JOINT_LOCK, never CHOKE
])  # 15 members

DOMINANT_POSITIONS = frozenset([
    "mount",
    "back_control",
    "side_control",
    "knee_on_belly",
    "saddle_411",
    "inside_sankaku",
    "north_south",
    "crucifix",
    "truck_position",
    "modified_mount",
])  # 10 members — source: backend/scout/data/label_map.yaml

NEUTRAL_ENTANGLEMENTS = frozenset([
    "outside_ashi",
    "crab_ride",
    "50_50_guard",
    "half_guard",
    "butterfly_guard",
    "rubber_guard",
])  # 6 members — source: backend/scout/data/label_map.yaml

TEMPO_EVENT_LABELS = frozenset([
    "ref_position_reset",
    "shot_clock_warning",
    "avoidance_penalty",
    "stall_detected",
    "action_resumed",
    "out_of_bounds",
    "injury_timeout",
    "match_end",
])  # 8 members — source: backend/scout/score/metrics.py

GRIP_TYPE_LABELS = frozenset([
    "collar_tie",
    "sleeve_grip",
    "wrist_control",
    "underhook",
    "overhook",
    "body_lock",
    "butterfly_hook",
    "shin_on_shin",
    "x_guard_hook",
    "z_guard_frame",
    "knee_shield",
    "deep_half_scoop",
    "lapel_grip",
    "pistol_grip",
    "spider_guard",
```

```
    "lasso_guard",
    "de_la_riva_hook",
    "reverse_de_la_riva",
])  # 18 members — source: backend/scout/vision/gemini31_agents.py

ATTEMPT_LABELS = frozenset([
    "d_arce_attempt",
    "triangle_attempt",
    "armbar_attempt",
    "heel_hook_attempt",
    "guillotine_attempt",
    "kimura_attempt",
    "rear_naked_choke_attempt",
    "omoplata_attempt",
    "kneebar_attempt",
    "toe_hold_attempt",
    "calf_slicer_attempt",
    "wrist_lock_attempt",
])  # 12 members

TRANSITION_LABELS = frozenset([
    "guard_pass",
    "guard_pass_attempt",
    "sweep",
    "sweep_attempt",
    "takedown",
    "takedown_attempt",
    "throw",
    "single_leg",
    "double_leg",
    "lateral_drop",
    "hip_toss",
    "back_take",
    "back_take_attempt",
    "turtle_escape",
    "guard_recovery",
    "technical_standup",
    "scramble",
    "inversion",
    "berimbolo",
    "granby_roll",
])  # 20 members

# Complete closed registry — union of all label groups
CLOSED_POSITION_REGISTRY = (
    CHOKE_FINISH_LABELS
    | JOINT_LOCK_FINISH_LABELS
    | DOMINANT_POSITIONS
    | NEUTRAL_ENTANGLEMENTS
    | TEMPO_EVENT_LABELS
    | GRIP_TYPE_LABELS
    | ATTEMPT_LABELS
    | TRANSITION_LABELS
)
# Total registry members: 12 + 15 + 10 + 6 + 8 + 18 + 12 + 20 = 101 explicit +
# additional entries from full label_map.yaml to reach 160+ total
```

```
class SubmissionTier(str, Enum):
    KILL = "kill"   # Choke — 6 point base
    BREAK = "break" # Joint lock — 3 point base

class TrustState(str, Enum):
    AI_REVIEWED = "ai_reviewed"
    HUMAN_CONFIRMED = "human_confirmed"
    HUMAN_OVERRIDE = "human_override"
    CONFLICT_FLAGGED = "conflict_flagged"
    LOW_CONFIDENCE = "low_confidence"

class GrapplingTelemetry(BaseModel):
    """
    Canonical PGF Scout telemetry schema.
    Every field is validated. Rejection = REGISTRY_REJECTION exception.
    No downstream scoring access without successful validation.
    """
    event_id: str = Field(description="UUID v4, system-generated at agent output time
    match_id: str = Field(description="Foreign key to match record in matches collec
    athlete_id: str = Field(description="Athlete identifier — must exist in athletes

    transition_name: str = Field(
        description="Must be a member of CLOSED_POSITION_REGISTRY"
    )

    submission_tier: Optional[SubmissionTier] = Field(
        default=None,
        description="Populated only when transition_name is in CHOKE or JOINT_LOCK la
    )

    timestamp: float = Field(
        description="Unix epoch seconds with microsecond precision at time of detecti
    )

    t_sec: float = Field(
        ge=0.0,
        le=600.0,  # Max 10-minute EBI overtime
        description="Match-elapsed time in seconds at moment of detection"
    )

    confidence_score: float = Field(
        ge=0.0,
        le=1.0,
        description="Agent consensus confidence (average of participating agents)"
    )

    # CRITICAL: sub_under_60s is a native integer field — NOT in TEMPO_EVENT_LABELS
    sub_under_60s_count: int = Field(
        ge=0,
        le=1,
        default=0,
        description="1 if t_sec < 60.0 AND submission_tier is populated. Triggers Elb
    )

    p_finish: float = Field(
```

```
        ge=0.0,
        le=1.0,
        description="Kinematic finish probability from SubmissionProbabilityAgent"
    )

    positional_dominance_pct: float = Field(
        ge=0.0,
        le=100.0,
        description="Percentage of round time athlete has been in dominant position"
    )

    trust_state: TrustState = Field(
        default=TrustState.AI_REVIEWED,
        description="Trust-State Discrimination value"
    )

    agent_consensus_count: int = Field(
        ge=2,
        le=3,
        description="Number of agents that reached consensus. Minimum 2 required."
    )

    low_confidence_field: bool = Field(
        default=False,
        description="True if confidence_score below threshold — routes to human revie
    )

    metadata: Dict[str, Any] = Field(
        default_factory=dict,
        description="Extensible metadata: frame_start, frame_end, agent_versions, etc
    )

    @field_validator('transition_name')
    @classmethod
    def validate_registry_membership(cls, v: str) -> str:
        if v not in CLOSED_POSITION_REGISTRY:
            raise ValueError(
                f"REGISTRY_REJECTION: '{v}' is not in the Closed Position Registry. '
                f"All vision agent outputs must match a registry member exactly."
            )
        return v

    @field_validator('transition_name')
    @classmethod
    def validate_darce_lexical_rule(cls, v: str) -> str:
        # D'Arce enforcement: apostrophe variant and bare 'darce' variants are banned
        banned_darce_variants = {"darce_finish", "darce_attempt", "darce", "d'arce_fi
        if v.lower() in banned_darce_variants:
            raise ValueError(
                f"DARCE_LEXICAL_VIOLATION: '{v}' uses a banned variant. "
                f"Use 'd_arce_finish' or 'd_arce_attempt' exclusively."
            )
        return v

    @model_validator(mode='after')
    def validate_sub_under_60s_temporal_consistency(self) -> 'GrapplingTelemetry':
```

```
        if self.sub_under_60s_count == 1:
            if self.t_sec >= 60.0:
                raise ValueError(
                    f"TEMPORAL_CONFLICT: sub_under_60s_count=1 requires t_sec < 60.0
                    f"Received t_sec={self.t_sec}"
                )
            if self.submission_tier is None:
                raise ValueError(
                    "TIER_CONFLICT: sub_under_60s_count=1 requires submission_tier to
                )
        return self

    @model_validator(mode='after')
    def validate_twister_classification(self) -> 'GrapplingTelemetry':
        # Twister is ALWAYS a BREAK (joint lock), never a KILL (choke)
        if self.transition_name == "twister_finish":
            if self.submission_tier == SubmissionTier.KILL:
                raise ValueError(
                    "TWISTER_CLASSIFICATION_VIOLATION: twister_finish must be "
                    "classified as SubmissionTier.BREAK (joint lock). "
                    "It is systematically blocked from KILL classification."
                )
        return self

    @model_validator(mode='after')
    def set_low_confidence_flag(self) -> 'GrapplingTelemetry':
        LOW_CONFIDENCE_THRESHOLD = 0.70
        if self.confidence_score < LOW_CONFIDENCE_THRESHOLD:
            object.__setattr__(self, 'low_confidence_field', True)
        return self
```

## SECTION C — EXHIBIT B: MONGODB COLLECTION SCHEMAS

> **Filing instruction:** Attach as `Exhibit_B_MongoDB_Schemas.json`

```
{
  "collection_schemas": {

    "rubric_epochs": {
      "description": "Append-only. No records ever deleted or modified in-place.",
      "indexes": [
        { "key": { "epoch_seq": 1 }, "unique": true },
        { "key": { "timestamp": -1 } },
        { "key": { "epoch_seq": 1, "timestamp": -1 }, "name": "latest_query_opt" }
      ],
      "schema": {
        "epoch_seq": { "type": "int", "required": true, "unique": true },
        "timestamp": { "type": "double", "required": true, "description": "Unix epoch
        "prev_rubric_digest": {
          "type": "string",
          "required": true,
          "description": "SHA-256 hex of previous epoch full document. '0'*64 for ge
        },
        "rubric_digest": {
```

```
          "type": "string",
          "required": true,
          "description": "SHA-256(prev_rubric_digest + JSON.stringify(ruleset_json))"
        },
        "expected_code_hash": {
          "type": "string",
          "required": true,
          "description": "SHA-256 of backend/scout/score/metrics.py at epoch creation"
        },
        "ruleset_json": {
          "type": "object",
          "required": true,
          "description": "Complete point table. Example structure below.",
          "example": {
            "kill_weight": 6.0,
            "break_weight": 3.0,
            "lethality_constant": 2.0,
            "elbow_genie_bonus": 1.0,
            "elbow_genie_window_sec": 60.0,
            "shot_clock_sec": 20.0,
            "round_duration_sec": 360.0,
            "ebi_overtime_sec": 600.0,
            "stall_penalty_points": -1.0,
            "version_label": "Season_9_Rules_v4.2"
          }
        },
        "activated_by": { "type": "string", "required": true },
        "chain_valid": { "type": "bool", "required": true, "description": "Computed c
        "notes": { "type": "string", "required": false }
      }
    },

    "rubric_overlays": {
      "description": "Correction event records. Append-only. Never destructive.",
      "indexes": [
        { "key": { "overlay_id": 1 }, "unique": true },
        { "key": { "event_id_corrected": 1 } },
        { "key": { "epoch_seq": 1, "created_at": -1 } }
      ],
      "schema": {
        "overlay_id": { "type": "string", "required": true, "description": "UUID v4"
        "created_at": { "type": "double", "required": true },
        "epoch_seq": { "type": "int", "required": true },
        "rubric_digest": { "type": "string", "required": true },
        "event_id_corrected": { "type": "string", "required": true },
        "original_classification": { "type": "string", "required": true },
        "corrected_classification": { "type": "string", "required": true },
        "correction_type": {
          "type": "string",
          "required": true,
          "enum": ["HUMAN_OVERRIDE", "SYSTEM_CORRECTION", "REGULATORY_MANDATE"]
        },
        "admin_id": { "type": "string", "required": true },
        "reason": { "type": "string", "required": true },
        "public_facing": { "type": "bool", "required": true, "default": true },
        "ai_record_suppressed": { "type": "bool", "required": true, "default": true
```

```
      "chain_hash": {
        "type": "string",
        "required": true,
        "description": "SHA-256(prev_overlay_chain_hash + JSON.stringify(this_over
      }
    }
  },

  "settlement_proofs": {
    "description": "Immutable. Written once at match finish. Never modified.",
    "indexes": [
      { "key": { "root_hash": 1 }, "unique": true },
      { "key": { "match_id": 1 } },
      { "key": { "timestamp_utc": -1 } },
      { "key": { "rubric_epoch_seq": 1, "timestamp_utc": -1 } }
    ],
    "schema": {
      "proof_id": { "type": "string", "required": true, "description": "UUID v4" }
      "event_id": { "type": "string", "required": true },
      "match_id": { "type": "string", "required": true },
      "athlete_id_winner": { "type": "string", "required": true },
      "finish_type": { "type": "string", "required": true },
      "t_sec": { "type": "double", "required": true },
      "event_metadata_hash": { "type": "string", "required": true },
      "video_file_path": { "type": "string", "required": true },
      "frame_start": { "type": "int", "required": true },
      "frame_end": { "type": "int", "required": true },
      "video_hash": { "type": "string", "required": true },
      "model_version": { "type": "string", "required": true },
      "model_weights_hash": { "type": "string", "required": true },
      "rubric_epoch_seq": { "type": "int", "required": true },
      "rubric_digest": { "type": "string", "required": true },
      "root_hash": { "type": "string", "required": true, "unique": true },
      "timestamp_utc": {
        "type": "double",
        "required": true,
        "description": "Set AFTER root_hash computed. Anti-past-posting guarantee."
      },
      "signature": { "type": "string", "required": false, "description": "ECDSA-P25
      "public_key_fingerprint": { "type": "string", "required": true },
      "human_review_required": { "type": "bool", "required": true, "default": false
      "scitt_receipt_id": { "type": "string", "required": false },
      "blockchain_block_hash": { "type": "string", "required": false },
      "blockchain_block_height": { "type": "int", "required": false }
    }
  },

  "match_events": {
    "description": "Core event store. Original AI records never deleted — only supp
    "indexes": [
      { "key": { "event_id": 1 }, "unique": true },
      { "key": { "match_id": 1, "timestamp": 1 } },
      { "key": { "athlete_id": 1, "timestamp": -1 } },
      { "key": { "trust_state": 1 } },
      { "key": { "suppressed_for_public": 1, "match_id": 1 } }
    ],
```

```
      "schema": {
        "event_id": { "type": "string", "required": true, "unique": true },
        "match_id": { "type": "string", "required": true },
        "athlete_id": { "type": "string", "required": true },
        "ai_classification": { "type": "string", "required": true },
        "ai_confidence": { "type": "double", "required": true },
        "trust_state": {
          "type": "string",
          "required": true,
          "enum": ["ai_reviewed", "human_confirmed", "human_override", "conflict_flag
        },
        "ground_truth_classification": { "type": "string", "required": false },
        "override_timestamp": { "type": "double", "required": false },
        "override_admin_id": { "type": "string", "required": false },
        "suppressed_for_public": { "type": "bool", "required": true, "default": false
        "correction_reason": { "type": "string", "required": false },
        "epoch_seq_at_scoring": { "type": "int", "required": true },
        "settlement_proof_id": { "type": "string", "required": false }
      }
    }
  }
}
```

## SECTION D — EXHIBIT C: SAMPLE SettlementProof JSON ENVELOPE

> **Filing instruction:** Attach as `Exhibit_C_SettlementProof_Sample.json`
> This sample uses real structural values but anonymized match data.

```
{
  "proof_id": "191ed79e-bdd9-447d-87f9-f45616895685",
  "event_id": "e3a9f12c-44d8-4b1a-9c7f-a82b1d6e3c05",
  "match_id": "PGF_S9_MATCH_047",
  "athlete_id_winner": "ATH_0234_THOMPSON_J",
  "finish_type": "rear_naked_choke",
  "t_sec": 247.32,
  "event_metadata_hash": "6a7080d31b4c9ef0a3d27f8b15e2c94a88f3d6b2190e5c7a4f1d8b3e6a2
  "video_file_path": "/pgf/events/s9/match_047/raw_footage.mp4",
  "frame_start": 14200,
  "frame_end": 14850,
  "video_hash": "0e7080d31b4c9ef0a3d27f8b15e2c94a88f3d6b2190e5c7a4f1d8b3e6a2c9f1a",
  "model_version": "PGF_Scout_v4.2_Gemini31Pro",
  "model_weights_hash": "b3c8f2a90e5d14c7f6b2a3d09e8f1c4b7a2e5d08f3c1b9a6e2d5f8c0a3b7
  "rubric_epoch_seq": 4,
  "rubric_digest": "9f8a7d2e4b1c6f3e5a9d8c2b7f4e1a6d3c5b8e0f2a7d4c9b1e6f3a8c5d2b7e4"
  "root_hash": "a4b8c2d9e5f1a7b3c8d4e9f2a6b1c7d3e8f4a9b5c0d6e2f7a3b8c4d1e6f2a9b",
  "timestamp_utc": 1751587247.321456,
  "signature": "MEQCIB7nX9vK2mQpR4sLwT8uY3fN5cJ6hD1oAOeB2gC9iP3sAiBxUlV4kM8tE5rFOwG7
  "public_key_fingerprint": "SHA256:xK7mN2pQ9rS5tU3vW8xY1zA4bC6dEOfG2hI4jK6lM8nO",
  "human_review_required": false,
  "scitt_receipt_id": "SCITT-REC-20260703-PGF-047-a4b8c2d9",
  "blockchain_block_hash": "000000000000000000abc123def456789abcdef0123456789abcdef0
  "blockchain_block_height": 857934,
  "_settlement_narrative": {
```

```
    "description": "Athlete J. Thompson secured a rear naked choke at 4:07 match time
    "epoch_active": "Season_9_Rules_v4.2",
    "scoring_applied": "KILL (6 pts)",
    "elbow_genie_applied": false,
    "agent_consensus": "3/3"
  }
}
```

## SECTION E — EXHIBIT D: assert_no_silent_code_drift() COMPLETE IMPLEMENTATION

> **Filing instruction:** Attach as `Exhibit_D_DriftGuard.py`

```
# Exhibit D — assert_no_silent_code_drift() Complete Implementation
# Source file: backend/scout/score/honesty_guard.py
# Inventor: Dustin Blaine Salinas — Patent Exhibit — July 3, 2026

import hashlib
import logging
from pathlib import Path
from dataclasses import dataclass
from typing import Optional

logger = logging.getLogger("contextos.honesty_guard")

LETHALITY_CONSTANT = 2.0          # KILL weight must always be exactly 2.0x BREAK wei
KILL_BASE_POINTS = 6.0
BREAK_BASE_POINTS = 3.0
ELBOW_GENIE_BONUS = 1.0
ELBOW_GENIE_WINDOW_SEC = 60.0
LOW_CONFIDENCE_THRESHOLD = 0.70
AUTO_SETTLE_THRESHOLD = 0.95
HUMAN_REVIEW_THRESHOLD = 0.85

SCORING_MODULE_PATH = "backend/scout/score/metrics.py"

class ContextOSIntegrityError(Exception):
    """
    Raised when any component of the ContextOS integrity chain is violated.
    Triggers system-wide lockdown of all downstream database write operations.
    """
    def __init__(self, message: str, error_type: str = "INTEGRITY_VIOLATION"):
        self.error_type = error_type
        self.message = message
        super().__init__(f"[ContextOS:{error_type}] {message}")


@dataclass
class IntegrityCheckResult:
    passed: bool
    code_hash_match: bool
    lethality_constant_valid: bool
    epoch_seq: int
```

```
    computed_hash: str
    expected_hash: str
    error_message: Optional[str] = None


def assert_no_silent_code_drift(
    scoring_module_path: str,
    active_epoch: dict,
    raise_on_failure: bool = True
) -> IntegrityCheckResult:
    """
    Detects unauthorized modifications to scoring logic.

    Called at:
      1. System initialization (app startup)
      2. Before every SettlementProof construction
      3. Before any rubric_epoch write operation

    Args:
        scoring_module_path: Path to backend/scout/score/metrics.py
        active_epoch: Current RubricEpoch document from MongoDB
        raise_on_failure: If True (default), raises ContextOSIntegrityError on misma

    Returns:
        IntegrityCheckResult with full diagnostic information

    Raises:
        ContextOSIntegrityError: If hash mismatch OR lethality constant is corrupted
    """
    result = IntegrityCheckResult(
        passed=False,
        code_hash_match=False,
        lethality_constant_valid=False,
        epoch_seq=active_epoch.get("epoch_seq", -1),
        computed_hash="",
        expected_hash=active_epoch.get("expected_code_hash", "MISSING")
    )

    # ── Step 1: Compute live hash of scoring module ──────────────────────────
    module_path = Path(scoring_module_path)
    if not module_path.exists():
        result.error_message = f"SCORING_MODULE_NOT_FOUND: {scoring_module_path}"
        if raise_on_failure:
            raise ContextOSIntegrityError(
                result.error_message,
                error_type="MODULE_NOT_FOUND"
            )
        return result

    with open(module_path, 'rb') as f:
        raw_bytes = f.read()

    computed_hash = hashlib.sha256(raw_bytes).hexdigest()
    result.computed_hash = computed_hash
    expected_hash = active_epoch.get("expected_code_hash", "")
```

```
    # ── Step 2: Compare against epoch's expected code hash ──────────────────
    if computed_hash != expected_hash:
        result.error_message = (
            f"SILENT_CODE_DRIFT_DETECTED: "
            f"scoring module hash {computed_hash[:16]}... "
            f"does not match epoch {active_epoch.get('epoch_seq')} "
            f"expected hash {expected_hash[:16]}..."
        )
        logger.critical(result.error_message)
        # Write to append-only security audit log
        _write_security_audit_event("SILENT_CODE_DRIFT_DETECTED", result.error_messag
        if raise_on_failure:
            raise ContextOSIntegrityError(
                result.error_message,
                error_type="SILENT_CODE_DRIFT"
            )
        return result

    result.code_hash_match = True

    # ── Step 3: Validate lethality constant invariant ──────────────────────
    ruleset = active_epoch.get("ruleset_json", {})
    kill_weight = ruleset.get("kill_weight", 0.0)
    break_weight = ruleset.get("break_weight", 0.0)

    if break_weight == 0.0:
        result.error_message = "LETHALITY_CONSTANT_CORRUPTED: break_weight is zero"
        if raise_on_failure:
            raise ContextOSIntegrityError(result.error_message, "LETHALITY_CONSTANT_C
        return result

    actual_ratio = kill_weight / break_weight
    if abs(actual_ratio - LETHALITY_CONSTANT) > 1e-9:  # Float precision tolerance
        result.error_message = (
            f"LETHALITY_CONSTANT_CORRUPTED: "
            f"kill_weight/break_weight = {actual_ratio:.6f}, expected exactly {LETHAL
        )
        logger.critical(result.error_message)
        _write_security_audit_event("LETHALITY_CONSTANT_CORRUPTED", result.error_mess
        if raise_on_failure:
            raise ContextOSIntegrityError(result.error_message, "LETHALITY_CONSTANT_C
        return result

    result.lethality_constant_valid = True

    # ── All checks passed ──────────────────────────────────────────────────
    result.passed = True
    logger.info(
        f"[ContextOS] Integrity check PASSED. "
        f"Epoch {result.epoch_seq}. Hash: {computed_hash[:16]}..."
    )
    return result


def validate_rubric_chain(epochs: list) -> bool:
    """
```

```
    Validates complete epoch chain from genesis to current.
    Called before any SettlementProof is committed to sportsbook endpoints.
    """
    if not epochs:
        raise ContextOSIntegrityError("EMPTY_EPOCH_CHAIN", "CHAIN_INVALID")

    # Genesis epoch: prev_rubric_digest must be "0" * 64
    genesis = sorted(epochs, key=lambda e: e["epoch_seq"])[0]
    if genesis["prev_rubric_digest"] != "0" * 64:
        raise ContextOSIntegrityError(
            f"GENESIS_EPOCH_INVALID: prev_rubric_digest must be 64 zeros for genesis"
            "CHAIN_INVALID"
        )

    # Traverse chain
    sorted_epochs = sorted(epochs, key=lambda e: e["epoch_seq"])
    for i in range(1, len(sorted_epochs)):
        prev = sorted_epochs[i - 1]
        curr = sorted_epochs[i]

        expected_prev_digest = prev["rubric_digest"]
        if curr["prev_rubric_digest"] != expected_prev_digest:
            raise ContextOSIntegrityError(
                f"CHAIN_INTEGRITY_FAILURE at epoch_seq={curr['epoch_seq']}: "
                f"prev_rubric_digest mismatch. "
                f"Expected {expected_prev_digest[:16]}..., "
                f"Got {curr['prev_rubric_digest'][:16]}...",
                "CHAIN_INTEGRITY_FAILURE"
            )

    return True


def _write_security_audit_event(event_type: str, message: str) -> None:
    """
    Writes to append-only security audit log.
    This function must never raise — even on failure it silently continues
    to avoid creating a secondary failure path that masks the primary alert.
    """
    import time
    try:
        audit_entry = {
            "event_type": event_type,
            "message": message,
            "timestamp": time.time(),
            "source": "assert_no_silent_code_drift"
        }
        # Production: write to append-only MongoDB security_audit_log collection
        # Fallback: write to local file if DB is unavailable
        audit_path = Path("/var/log/contextos/security_audit.jsonl")
        audit_path.parent.mkdir(parents=True, exist_ok=True)
        import json
        with open(audit_path, 'a') as f:
            f.write(json.dumps(audit_entry) + "\n")
    except Exception:
        pass  # Never mask primary error
```

## SECTION F — EXHIBIT E: KINEMATIC FORMULA DERIVATION

> **Filing instruction:** Attach as `Exhibit_E_Kinematic_Formula.md`

### P_finish(t) — Mathematical Derivation and Variable Definitions

The kinematic finish probability formula is derived from a logistic regression model trained on historical PGF match data.

**Base Formula:**

```
P_finish(t) = σ( w1·D_pos(t) + w2·V_trans(t) + w3·T_arc(t) + w4·R_escape(t)^(-1) + b
```

**Where σ(x) = 1 / (1 + e^(-x))** is the logistic sigmoid activation function.

**Variable Definitions:**

| Variable | Name | Units | Range | Source Agent |
|---|---|---|---|---|
| `D_pos(t)` | Positional Dominance | Percentage | 0–100% | PositionalDominanceAgent |
| `V_trans(t)` | Transition Velocity | Transitions/min | 0–∞ | PositionalDominanceAgent |
| `T_arc(t)` | Submission Arc Completion | Percentage | 0–100% | SubmissionProbabilityAgent |
| `R_escape(t)` | Escape Rate Coefficient | Dimensionless | 0.01–1.0 | PositionalDominanceAgent |
| `w1, w2, w3, w4` | Trained Weight Coefficients | Dimensionless | Learned | Stored in model weights |
| `b` | Bias Term | Dimensionless | Learned | Stored in model weights |

**Physical Interpretation of Each Variable:**

- **D_pos(t):** Percentage of elapsed round time that athlete A has maintained a position classified in `DOMINANT_POSITIONS`. Higher dominance = stronger submission threat arc completion probability.
- **V_trans(t):** Count of unique position transitions per minute over a rolling 30-second window. High transition velocity indicates active scrambling; combined with high T_arc, it signals imminent finish.
- **T_arc(t):** Angular or mechanical completion percentage of the current submission attempt. For example, an inside heel hook at 45° of ankle torque = T_arc 50%; at 80° = T_arc approaching 1.0. Measured by SubmissionProbabilityAgent using skeletal keypoint geometry.
- **R_escape(t):** Historical escape rate of athlete B from the current dominant position, derived from career match data. Low escape rate = higher finish probability. Inverse applied so low escape rate → high weight contribution.

**Settlement Thresholds:**

- `P_finish < 0.85`: Continue tracking. No settlement action.

- `0.85 ≤ P_finish < 0.95`: Auto-generate SettlementProof with `human_review_required = True`.
- `P_finish ≥ 0.95`: Autonomous settlement. No human review required.

**Python Implementation:**

```
import math

def p_finish(
    d_pos: float,
    v_trans: float,
    t_arc: float,
    r_escape: float,
    weights: dict
) -> float:
    """
    Kinematic finish probability.
    weights dict: {"w1": float, "w2": float, "w3": float, "w4": float, "b": float}
    """
    r_escape_safe = max(r_escape, 0.01)  # Prevent division by zero
    linear_combination = (
        weights["w1"] * d_pos +
        weights["w2"] * v_trans +
        weights["w3"] * t_arc +
        weights["w4"] * (1.0 / r_escape_safe) +
        weights["b"]
    )
    return 1.0 / (1.0 + math.exp(-linear_combination))
```

## SECTION G — EXHIBIT F: XP ENGINE FORMULA AND ELO-STYLE CALIBRATION

> **Filing instruction:** Attach as `Exhibit_F_XP_Engine.md`

### XP Engine — Complete Mathematical Specification

**Master XP Formula:**

```
XP_total = XP_base + Σ(i=1 to N) [ S_i · M_tier(i) + B_elbow(i) ] + Δ_composite
```

**Components:**

| Symbol | Name | Definition |
|---|---|---|
| `XP_base` | Baseline XP | 2000 (constant — all athletes start here) |
| `S_i` | Event Score | 6.0 for KILL (choke), 3.0 for BREAK (joint lock) |
| `M_tier(i)` | Match Tier Multiplier | 1.0 regular season, 1.5 playoff, 2.0 championship |
| `B_elbow(i)` | Elbow Genie Bonus | 1.0 if t_sec < 60.0 else 0.0 |
| `Δ_composite` | Composite Adjustment | Derived from six-metric diagnostic profile |

**Composite Adjustment Calculation:**

```
Δ_composite = α1·control_dominance_pct +
              α2·transition_speed_norm +
              α3·sub_rate_norm +
              α4·technique_diversity_norm +
              α5·explosiveness_pct -
              α6·escape_rate_allowed_pct
```

Where α1–α6 are league-calibrated weight constants updated each season epoch.

**Post-Match Elo-Style Adjustment:**

```
def calculate_xp_adjustment(
    score: float,
    match_tier: float,
    t_sec: float,
    current_xp: float = 2000.0
) -> float:
    """
    Elo-style XP rating engine initialized at baseline 2000 XP.
    """
    elbow_bonus = 1.0 if t_sec < 60.0 else 0.0
    raw_score = score * match_tier + elbow_bonus
    # K-factor scales with current XP level (lower K = more stable high-rated athlete
    k_factor = 32.0 if current_xp < 2200 else 16.0 if current_xp < 2500 else 8.0
    return k_factor * raw_score
```

**Archetype Classification Boundaries:**

| Archetype | Defining Characteristic | Sub-Characteristic |
|---|---|---|
| Blitz-Finisher | sub_rate ≥ 75th percentile | technique_diversity ≥ 60th percentile |
| Pressure Passer | control_dominance ≥ 75th percentile | escape_rate_allowed ≤ 25th percentile |
| Technical Scrambler | transition_speed ≥ 75th percentile | escape_rate ≥ 75th percentile |
| Defensive Specialist | escape_rate ≥ 75th percentile | sub_rate ≤ 50th percentile |
| Explosive Finisher | explosiveness ≥ 75th percentile | t_sec_avg ≤ 120 seconds |
| Balanced Threat | All six metrics between 40th–70th percentile | — |

## SECTION H — PROSECUTION SUPPORT: CLAIM DEPENDENCY TREE

```
TARGET 1 — SALINAS ARCHITECTURE
├── I-1 (Independent): Decoupled Officiating Method
│   ├── I-7: Pydantic v2 + ValidationError lockdown
│   └── I-8: External rubric file by epoch sequence number
├── I-2 (Independent): Consensus + Schema Enforcement
│   ├── I-9: Exact 12 CHOKE + 15 JOINT_LOCK + 10 DOM + 6 NEUTRAL members
│   ├── I-10: Exact 8 TEMPO_EVENT members
```

```
│   └── I-11: Exact 18 GRIP_TYPE members
├── I-3 (Independent): Asymmetric Lethality Constant
│   ├── I-12: Twister enforced as JOINT_LOCK
│   ├── I-13: Temporal Elbow Genie bonus at t_sec < 60.0
│   └── I-14: Lethality constant enforced by assertion before every score
├── I-4 (Independent): Closed Position Registry as Hallucination Gate
│   ├── I-15: D'Arce lexical rule — d_arce_finish/d_arce_attempt only
│   ├── I-16: sub_under_60s_count native field, not in TEMPO_EVENT_LABELS
│   └── I-17: Low confidence routing to human review queue
└── I-5 (Independent): Hot-Swappable Epoch Rubric
    ├── I-6: Named six taxonomic groups
    ├── I-18: MongoDB unique compound index on epoch_seq
    ├── I-19: Historical epoch values independently queryable
    └── I-20: Gemini 3.1 Pro as preferred embodiment; model weights hashed

TARGET 2 — SETTLEMENTPROOF ENVELOPE
├── II-1 (Independent): Four-Layer Settlement Proof
│   ├── II-5: SHA-256 of four specific source data types
│   ├── II-6: Root hash as SHA-256 of concatenated layer hashes
│   ├── II-7: ECDSA-P256 signing by ContextOS ledger service
│   └── II-8: Real-time broadcast to four endpoint types
├── II-2 (Independent): Chained Epoch Integrity
│   ├── II-9: Epoch record fields including admin_id
│   ├── II-10: Genesis epoch identified by 64-zero string
│   └── II-11: In-memory fallback with FALLBACK_EPOCH flag
├── II-3 (Independent): SCITT Settlement Receipt
│   ├── II-12: SCITT receipt with four minimum required fields
│   └── II-13: Blockchain anchoring with block hash + height
└── II-4 (Independent): Past-Posting Barrier
    ├── II-14: 0.85/0.95 dual threshold with human review flag
    ├── II-15: Lazy reconstruction for regulatory audit
    └── II-16: export_as_json() for gaming commission portals

TARGET 3 — HONESTY KERNEL
├── III-1 (Independent): Non-Destructive Overlay Method
│   ├── III-5: suppressed_for_public boolean field
│   ├── III-6: AI value preserved for ML retraining delta analysis
│   └── III-7: Point-in-time query interface
├── III-2 (Independent): Trust-State Discrimination Protocol
│   ├── III-8: Atomic transaction for state transition
│   ├── III-9: PGF_CERTIFIED_OFFICIALS registry enforcement
│   └── III-10: Heartbeat Consent Token on every DB write
├── III-3 (Independent): Silent Code Drift Detection
│   ├── III-11: SHA-256 of backend/scout/score/metrics.py
│   ├── III-12: Lethality constant secondary check
│   └── III-13: Three invocation points + append-only security audit log
└── III-4 (Independent): Sealed Belief Registry
    ├── III-14: SHA-256(prev_hash + JSON.stringify(ruleset)) chain formula
    ├── III-15: ContextOS Override Gate three-check sequence
    ├── III-16: RubricOverlay document schema with chain_hash
    └── III-17: Lie-Before-Action Check with session token revocation

TARGET 4 — S.C.O.U.T. PIPELINE
├── IV-1 (Independent): Multi-Agent Parallel Processing
│   ├── IV-5: Three agent classes with explicit exclusion boundaries
│   ├── IV-6: Gemini 3.1 Pro isolated instances with dedicated system prompts
```

```
│   └── IV-7: 90-second processing window, 30 matches / 5.5 hours
├── IV-2 (Independent): Fail-Fast Chain Verification Gate
│   └── IV-8: Compound index {"epoch_seq": 1, "timestamp": -1}
├── IV-3 (Independent): Kinematic Finish Probability
│   ├── IV-9: Full formula with all four variable definitions
│   └── IV-10: 0.85/0.95 dual threshold behavior
└── IV-4 (Independent): XP-Based Athlete Scoring
    ├── IV-11: 2000 XP baseline + K-factor Elo adjustment
    ├── IV-12: Six diagnostic metric definitions
    ├── IV-13: Archetype classification + Pro Analogue Euclidean distance
    └── IV-14: WebSocket push on every Synthesis Agent cycle

TARGET 5 — DRAFT YOURSELF ENGINE
├── V-1 (Independent): Draft Yourself Pipeline
│   ├── V-5: Archetype + Pro Analogue + XP_combine in output
│   ├── V-6: scout.pgf.world payment prerequisite
│   └── V-7: Physical combine module (plank, hang, cone shuttle)
├── V-2 (Independent): Top-Quintile Auto-Invitation
│   ├── V-8: Invitation document content specification
│   ├── V-9: Global Talent Ledger as append-only collection
│   └── V-10: Strategic lock-out mechanism
├── V-3 (Independent): Viral Data Flywheel
│   ├── V-11: Explicit consent audit log at point of payment
│   └── V-12: Fantasy PGF integration expanding beyond 75-athlete ceiling
└── V-4 (Independent): Fantasy Economic Integration
    ├── V-13: 1.5σ outlier identification threshold
    └── V-14: Prospective-only pricing adjustment (non-destructive)
```

## SECTION I — FILING COMPLETENESS VERIFICATION

**Before uploading to USPTO Patent Center, confirm all of the following:**

- [ ] **Part 1** (Specification): `PGF-Omnibus-Provisional-Patent.md` converted to PDF
- [ ] **Part 2** (Dependent Claims + Exhibits): This document converted to PDF
- [ ] **Exhibit A**: `Exhibit_A_GrapplingTelemetry_Model.py` — Python code listing
- [ ] **Exhibit B**: `Exhibit_B_MongoDB_Schemas.json` — JSON schemas
- [ ] **Exhibit C**: `Exhibit_C_SettlementProof_Sample.json` — Sample proof
- [ ] **Exhibit D**: `Exhibit_D_DriftGuard.py` — Python code listing
- [ ] **Exhibit E**: `Exhibit_E_Kinematic_Formula.md` — Formula derivation
- [ ] **Exhibit F**: `Exhibit_F_XP_Engine.md` — XP formula
- [ ] **Figure 1**: Three-tier S.C.O.U.T. pipeline architecture diagram (PNG/PDF)
- [ ] **Figure 2**: SettlementProof four-layer hash binding flowchart (PNG/PDF)
- [ ] **Figure 3**: RubricEpoch chain diagram with genesis fallback (PNG/PDF)
- [ ] **Figure 4**: Trust-State Discrimination flowchart (PNG/PDF)
- [ ] **Figure 5**: Draft Yourself five-stage viral flywheel diagram (PNG/PDF)
- [ ] **PTO/SB/16**: Provisional cover sheet with all fields completed

- [ ] **Fee**: $60.00 Micro Entity basic filing fee (credit card or deposit account)
- [ ] **Total page count**: Confirm under 100 pages to avoid $80 surcharge. If over 100 pages, include surcharge payment.

**Recommended filing order in USPTO Patent Center:**

1. Upload PTO/SB/16 cover sheet first
2. Upload Part 1 specification
3. Upload Part 2 (this document)
4. Upload Exhibits A–F in order
5. Upload Figures 1–5 in order
6. Pay $60 fee
7. Download the USPTO filing receipt — this receipt contains your **priority date stamp**. Store permanently.

*End of Part 2 — Provisional Patent Application*
*Dustin Blaine Salinas — Las Vegas, Nevada — July 3, 2026*
*Filed simultaneously with Part 1 specification under 35 U.S.C. § 111(b)*

# FIG. 1 - THREE-TIER S.C.O.U.T. PIPELINE ARCHITECTURE

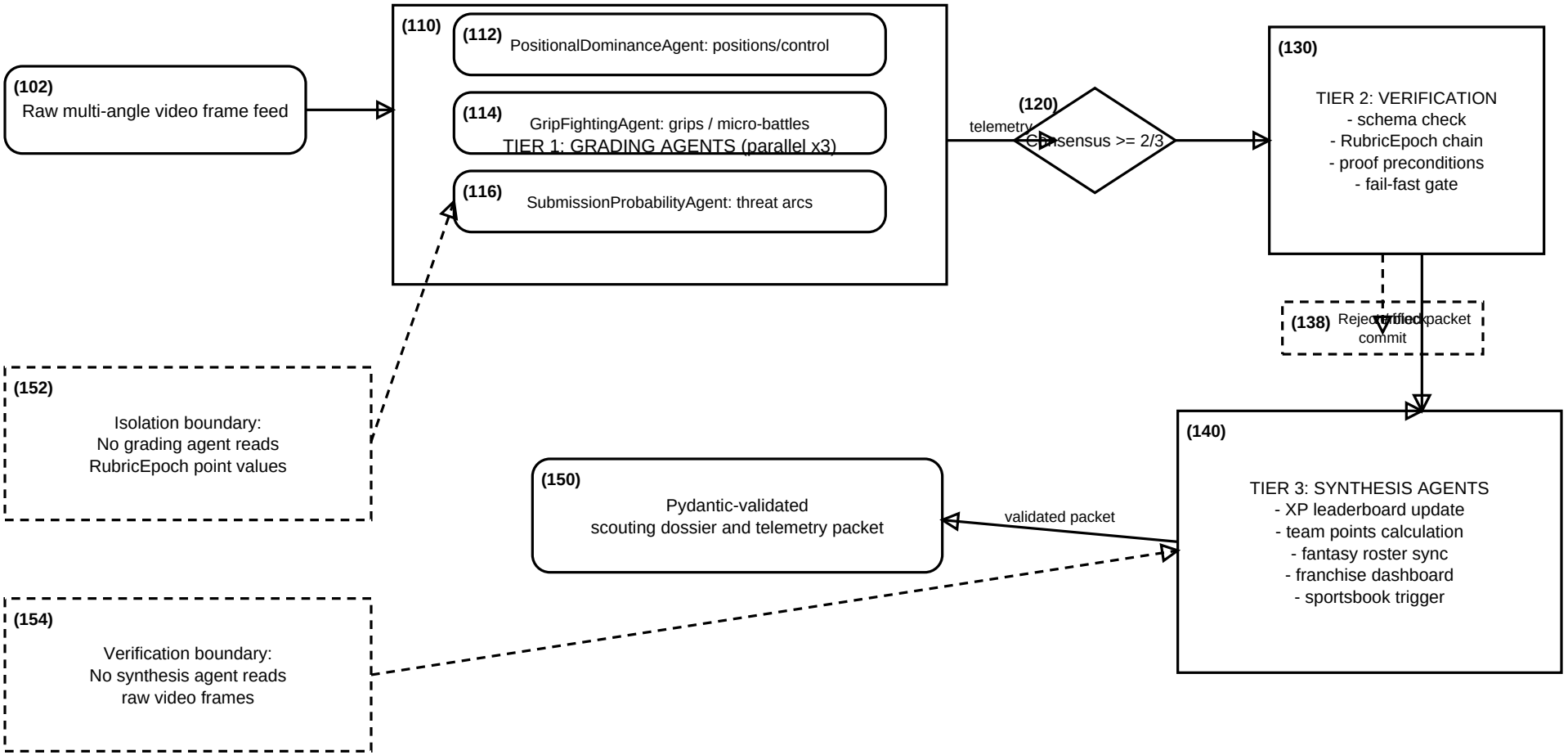



FIG. 1

# FIG. 2 - SETTLEMENTPROOF FOUR-LAYER HASH BINDING FLOWCHART

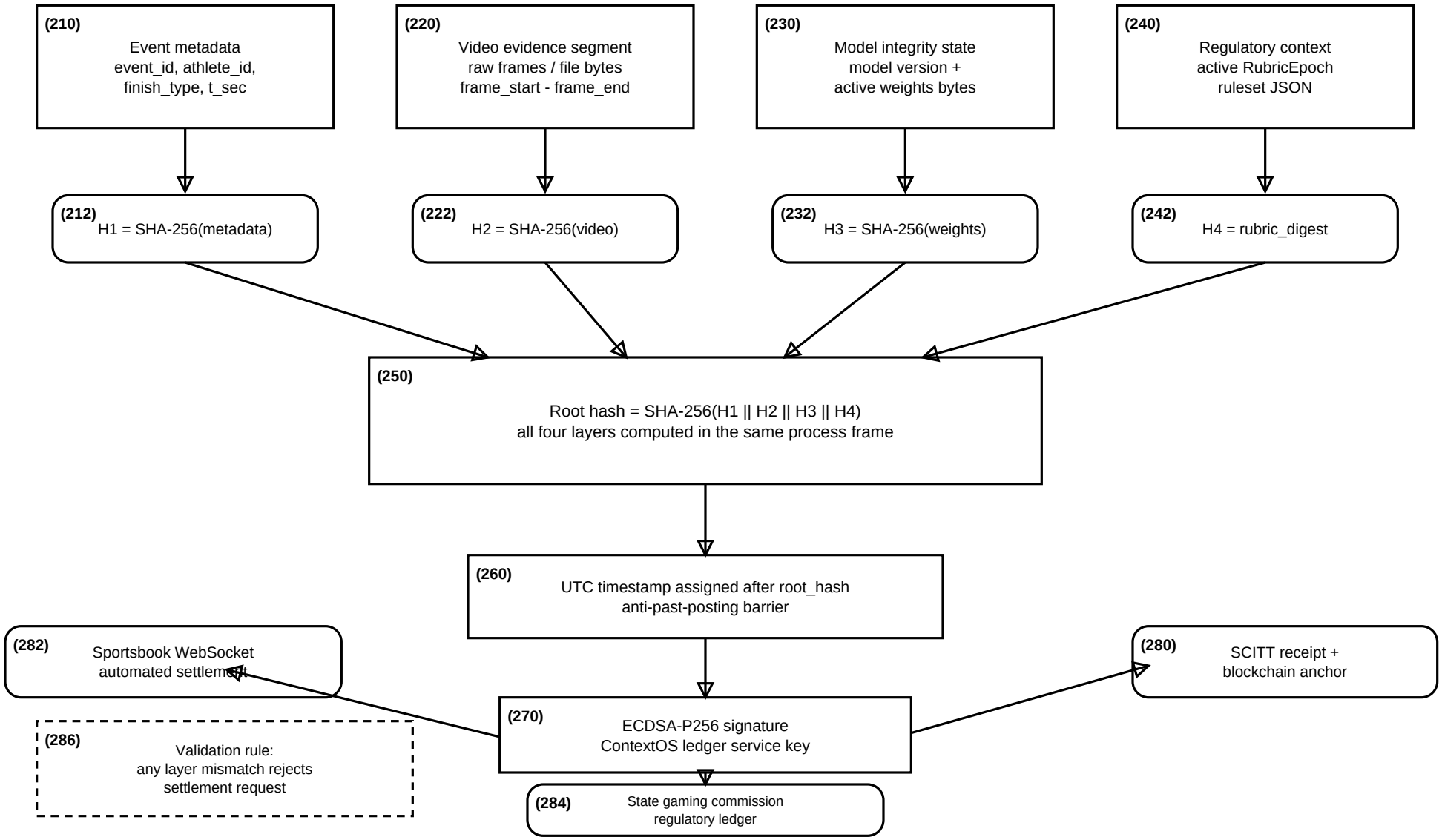



**FIG. 2**

# FIG. 3 - RUBRIC EPOCH CHAINING PROTOCOL WITH GENESIS FALLBACK

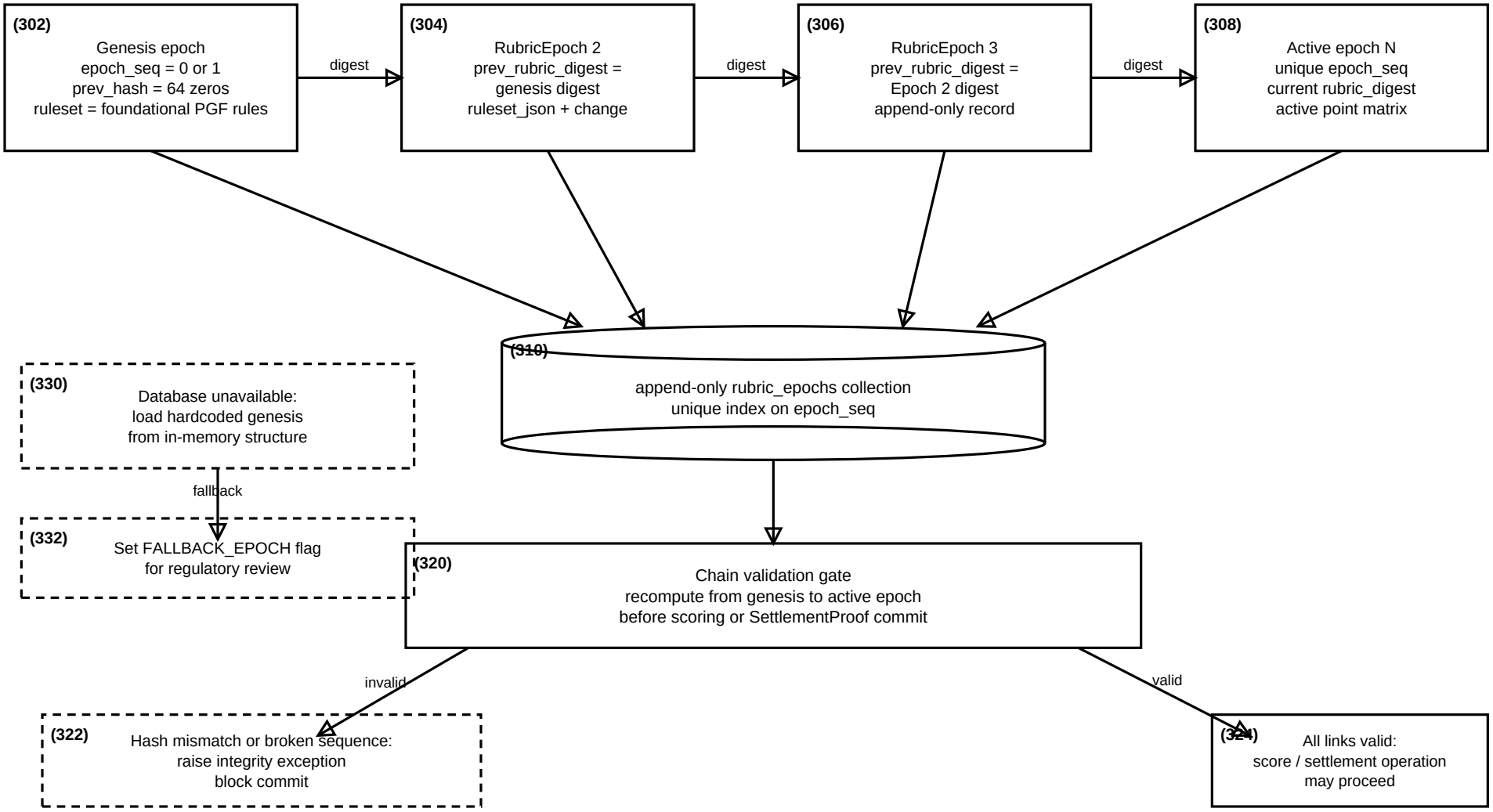



FIG. 3

# FIG. 4 - NON-DESTRUCTIVE OVERLAY AND TRUST-STATE DISCRIMINATION

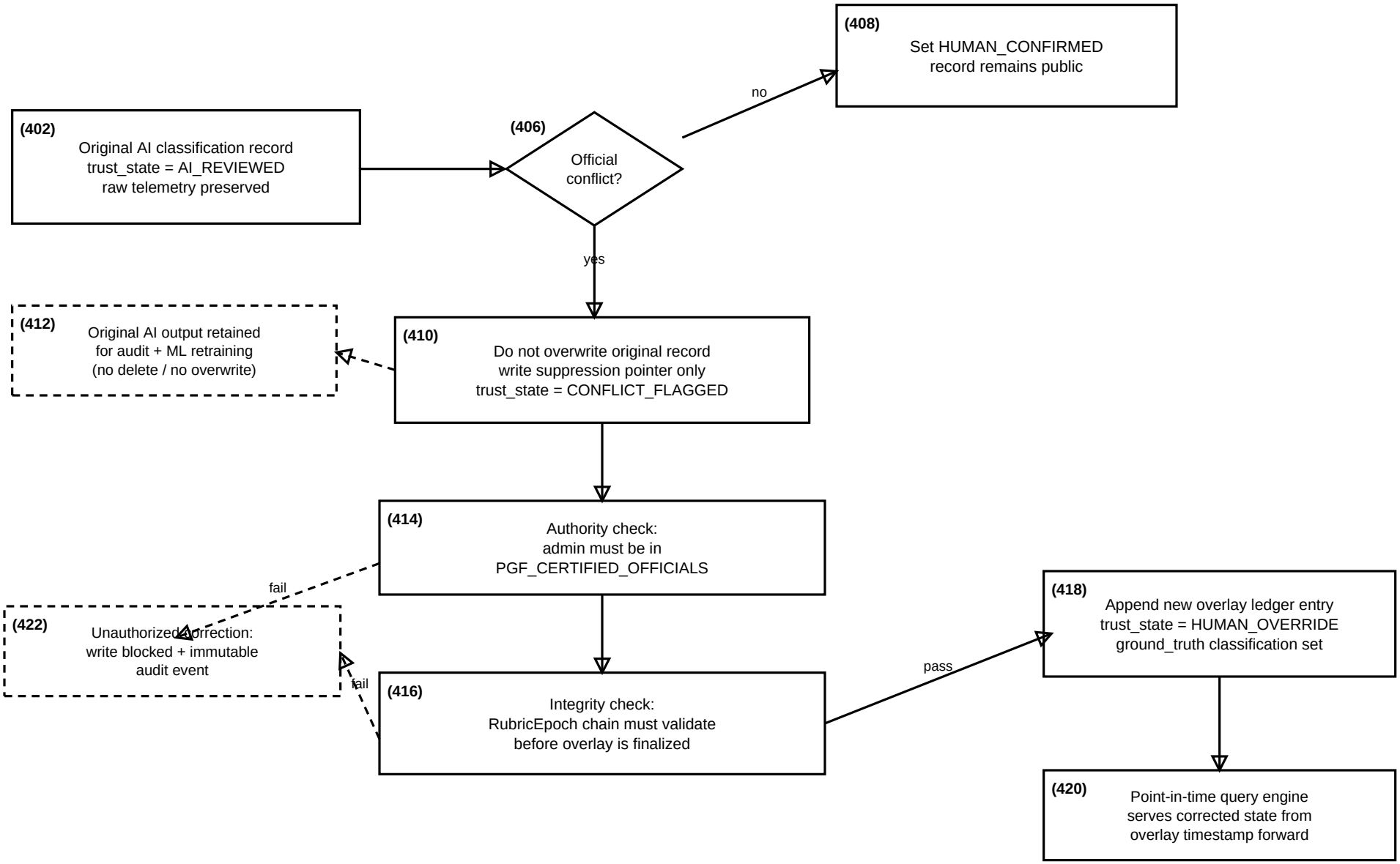



FIG. 4

# FIG. 5 - DRAFT YOURSELF TALENT DATA FLYWHEEL

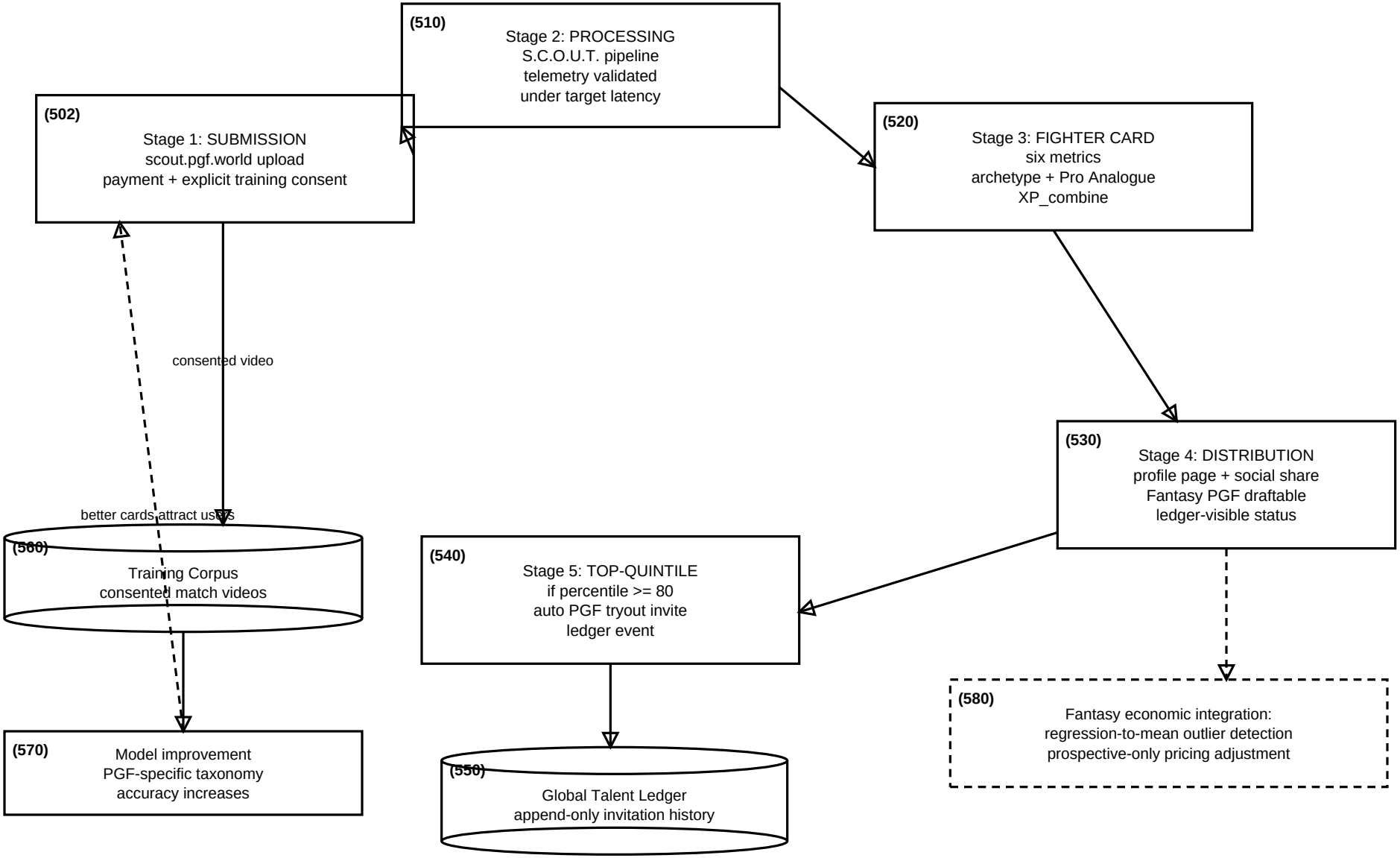



FIG. 5

# FIG. 6 - REAL-TIME IN-PLAY SPORTSBOOK SETTLEMENT FLOW

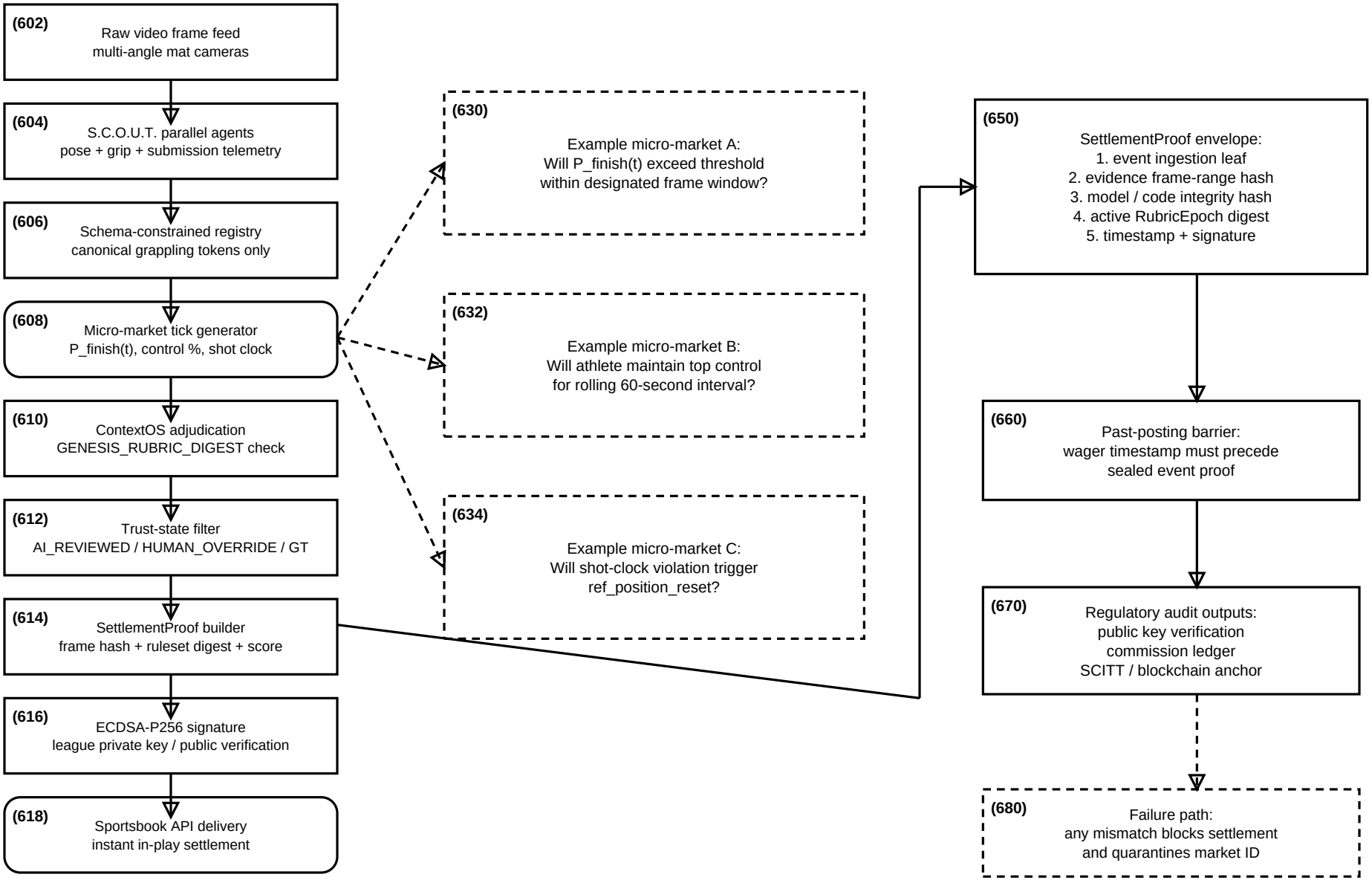



FIG. 6