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)