# Technical Specification: Unified Multi-Code Bet Generation Engine

This document provides the detailed technical specifications, formulas, and data schemas required to build the application.

### **1. Core Quantitative Formulas**

#### **1.2 Value Score Calculation**

The Value Score is a measure of the expected value of a bet, normalized by the market price.

* **Formula:** Value\_Score = (model\_probability / fair\_implied\_probability) - 1
* model\_probability: The probability of an outcome as determined by our internal predictive model.
* fair\_implied\_probability: The bookmaker's decimal odds converted to a probability, with the overround (vig) removed.

#### **1.4.1 Fractional Kelly Criterion**

The Fractional Kelly Criterion is used for stake sizing to balance growth and risk.

* **Formula:** Stake = Bankroll \* Kelly\_Fraction \* ((Decimal\_Odds \* Probability) - 1) / (Decimal\_Odds - 1)
* Kelly\_Fraction: A user-defined fraction (e.g., 0.5) to reduce risk.
* max\_stake\_cap: A hard cap on any single bet as a percentage of bankroll (e.g., 4%).

#### **3.3 Dynamic Fractional Kelly Staking**

This evolves the staking formula to incorporate a Model Confidence Score (MCS).

* **Formula:** Dynamic\_Kelly\_Fraction = Base\_Kelly\_Fraction \* MCS
* The Dynamic\_Kelly\_Fraction then replaces the static Kelly\_Fraction in the staking formula.
* **Test Case:** A bet with MCS=0.9 should have a recommended stake significantly larger than an identical bet with MCS=0.4.

### **2. Data Schemas (Pydantic)**

#### **Table 2.1: UnifiedRacingData Schema**

| Field | Data Type | Description |
| --- | --- | --- |
| event\_id | String | Unique identifier for the event. |
| race\_id | String | Unique identifier for the race. |
| race\_name | String | Name of the race. |
| venue | String | Racetrack venue. |
| race\_start\_time | DateTime | Official start time of the race. |
| runners | List | A list of runner objects. |

**Nested Runner Object:**

| Field | Data Type | Description |
| --- | --- | --- |
| runner\_id | String | Unique ID for the runner. |
| runner\_name | String | Name of the horse. |
| barrier | Integer | Starting barrier number. |
| win\_odds | Float | Decimal odds for winning. |
| gear\_changes | String | Description of any gear changes. |
| sectional\_times | List[Float] | List of sectional timing data. |

#### **Table 2.2: UnifiedSportsData Schema**

| Field | Data Type | Description |
| --- | --- | --- |
| event\_id | String | Unique identifier for the event. |
| sport\_key | String | Key for the sport (e.g., 'aussierules\_afl'). |
| home\_team | String | Name of the home team. |
| away\_team | String | Name of the away team. |
| event\_start\_time | DateTime | Official start time of the game. |
| markets | List[Market] | A list of market objects. |

**Nested Market Object:**

| Field | Data Type | Description |
| --- | --- | --- |
| market\_key | String | Key for the market (e.g., 'h2h', 'player\_tries'). |
| outcomes | List[Outcome] | A list of outcome objects. |

**Nested Outcome Object:**

| Field | Data Type | Description |
| --- | --- | --- |
| name | String | Name of the outcome (e.g., team name, player name). |
| price | Float | Decimal odds for the outcome. |
| prop\_line | Float | The line for player prop bets (e.g., 0.5). |

### **5. Advanced Logic Specifications**

#### **5.1 Advanced Dependence Modeling (Student's t-Copula)**

To price Same-Game Multis, a Student's t-Copula will be used to model the tail dependence between correlated events.

**Table 3: Advanced Logic Decomposition Map (Student's t-Copula)**

| Logical Step | Function/Method to Create | Specific Objective |
| --- | --- | --- |
| 1. Create Class Structure | CopulaCorrelationEngine class | Create a class to encapsulate the copula logic. |
| 2. Fit Copula to Data | fit(self, historical\_data) | Use the copulas library to fit a bivariate student\_t copula to historical outcome data. |
| 3. Calculate Conditional Probability | get\_conditional\_prob(self, u, v) | Use the fitted copula's conditional distribution function, `C(v |
| 4. Price the SGM | price\_sgm(self, prob\_A, prob\_B) | Orchestrate the process: transform probabilities, get conditional probability, and return the final joint probability `prob\_A \* P(B |