Perception vs. Performance: Quantifying Decision Accuracy in the College Football Playoff

Quinton Peters
January 16th, 2025

Abstract

The College Football Playoff (CFP) is a centerpiece of college football, tasked with determining the top teams in the sport. However, its selection process is influenced by both preseason rankings and ongoing perceptions of team strength throughout the season. This study investigates the impact of these perceptions—whether rooted in preseason biases or week-to-week narratives—on the accuracy and fairness of the CFP's expanded 12-team format. Through quantitative simulations of different ranking systems and committee decision-making styles, we assess how these perceptions influence rankings, stability, and outcomes. The analysis highlights the trade-offs between rewarding performance on the field and accounting for perceived strength, providing insights into how perceptions shape the college football landscape.

Contents

1 Introduction

The College Football Playoff (CFP) system, introduced in 2014, revolutionized how college football determines its champion by replacing the Bowl Championship Series (BCS) with a selection process driven by a committee. This committee, composed of athletic directors, coaches, and former players, meets weekly during the season to rank teams based on a blend of quantitative metrics and subjective evaluations. Their criteria include game results, strength of schedule, head-to-head outcomes, conference championships, and other contextual factors (such as the availability of key players and coaches). While this human-driven approach adds flexibility and judgment to the ranking process, it also invites controversy over potential biases.

In the 2024 season, this controversy reached new heights when multiple teams from the Southeastern Conference (SEC) with higher perceived strength but multiple losses were left out of the College Football Playoff while teams with fewer losses from less heralded conferences were put in the CFP. Critics argued that the CFP committee ignored and undervalued the historical prestige of those teams and their conference's elite reputation/competition, while supporters maintained that the the on field play of those teams justified their rankings. This debate epitomizes the tension between rewarding perceived strength and recognizing on-field results.

The controversy is further compounded by the expanded 12-team playoff format, which adds more opportunities for teams to compete but also increases the stakes for those on the playoff bubble. A fundamental question emerges: should teams be ranked based on their perceived strength, shaped by preseason expectations and historical dominance, or should rankings reflect purely on-field performance? Answering this question is critical to ensuring the CFP fulfills its promise of fairness and transparency.

This paper explores these issues through a quantitative lens, using simulations to evaluate how systemic biases, shaped by both preseason and week-to-week perceptions, impact team rankings. By examining the trade-offs between stability, accuracy, and excitement in rankings, we aim to provide insights into the dynamics of the CFP system and its broader implications for the sport.

2 Systemic Biases in Perceived Strength

Perceptions of team strength, whether rooted in preseason rankings, historical dominance, or week-to-week narratives, play a pivotal role in shaping the trajectory of a college football season. While these perceptions can provide valuable context for properly evaluating teams, they can also introduce systemic biases that perpetuate rankings and playoff opportunities throughout the remainder of the season. This section explores how such biases arise and examines both the arguments for and against their influence.

2.1 The Role of Preseason Rankings

Preseason rankings are intended to offer an informed baseline for evaluating teams before the season begins. Supporters argue that these rankings are valuable because they aggregate expert opinions, historical data, and predictive metrics. They serve as a starting point for fans and analysts to engage with the season, creating excitement and narratives that drive interest in the sport.

However, critics contend that preseason rankings are inherently flawed because they are based on incomplete information. Teams evolve significantly year to year due to player turnover, coaching changes, and other factors, making it nearly impossible to accurately assess team strength before any games are played. Furthermore, high preseason rankings can create "ranking inertia," where teams retain their positions even with comparable or inferior on-field performance to lower-ranked teams. This inertia disproportionately benefits historically strong programs and penalizes teams starting the season unranked, as they must overcome both on-field challenges and perception gaps to gain recognition.

Despite these criticisms, preseason rankings also have a stabilizing effect on the ranking system. Without an initial framework, weekly rankings might be more volatile, leading to overreactions to early-season results. For example, a single upset could have an outsized influence on rankings in the absence of preseason expectations, creating chaos in the playoff selection process.

2.2 Week-to-Week Biases in Team Evaluation

As the season progresses, week-to-week biases influence how teams are evaluated based on their performances. Teams with strong reputations or high preseason rankings often receive the benefit of the doubt in close games or unexpected losses. Supporters argue that these biases are not inherently negative; rather, they reflect the committee's nuanced understanding of context. For instance, a narrow loss to a top-ranked opponent might demonstrate a team's strength, whereas a similar loss to an unranked opponent could indicate vulnerability.

Critics, however, argue that week-to-week biases can lead to inconsistent evaluations. Teams from less prominent conferences may face harsher scrutiny, as their victories are often discounted due to perceived weaker competition. For example, a mid-major team with a dominant undefeated record might struggle to crack the top rankings, while a team from a power conference with multiple losses remains in contention due to the perceived strength of its schedule.

The role of media and public narratives further complicates week-to-week evaluations. High-profile programs often dominate coverage, shaping public perception and potentially influencing committee decisions. This creates a feedback loop where already prominent teams benefit from increased visibility and favorable interpretations of their performances. However, proponents argue that media-driven narratives also spotlight deserving teams, bringing attention to exceptional performances that might otherwise be overlooked.

2.3 Impact on Playoff Opportunities

Biases in rankings have direct implications for playoff opportunities, affecting not only which teams are selected but also how they are seeded. Teams with higher rankings gain access to more favorable matchups, marquee games, and increased media exposure, further bolstering their reputations. Proponents of this system argue that rewarding teams perceived as strong incentivizes programs to schedule tougher opponents, thereby elevating the overall quality of college football.

On the other hand, critics highlight the barriers faced by teams from smaller conferences or with weaker perceived schedules. These teams often need to achieve near-perfect seasons to receive consideration, as their victories are devalued relative to those of power-conference teams. This creates a systemic disadvantage, where teams must not only perform well but also overcome entrenched perceptions to compete for playoff spots.

Supporters of the current system argue that perception-based evaluations are necessary to account for differences in schedule difficulty and conference strength. Without such considerations, rankings might overvalue teams that dominate weaker competition while undervaluing those that perform well against tougher opponents. Critics counter that these subjective adjustments often exaggerate conference disparities, leading to an uneven playing field.

3 Evaluating the Impact of Preseason Rankings

As shown, preseason rankings play a pivotal role in shaping the narrative of a college football season. These rankings, published before any games are played, significantly influence perceptions of team strength and the value of later wins and losses. In this section, we examine the mechanisms through which preseason rankings impact a team's season, and we simulate a scenario to evaluate the consequences of an entirely incorrect (inverse) preseason poll.

3.1 The Inverse Ranking Scenario: Small Model

To assess the impact of preseason ranking bias, we simulate a hypothetical scenario in which the initial College Football Playoff (CFP) rankings are completely incorrect, placing the best team as the worst and vice versa. This analysis explores how such an error propagates through the season and affects both rankings and perceptions of team strength. However, we must first understand the general qualitative trends to look for, so we start with a smaller, subsidized model that can show us the short term responses we will need to draw final conclusions.

3.1.1 Code Overview

The simulation is based on Python code that models a five-week college football season for 26 teams. Key elements of the code include:

- **Team Generation:** Teams are assigned a `true_rank` (1 = best, 26 = worst) and an `cfp_rank` (inverse of `true_rank`).
- **Game Outcomes:** The probability of a team winning is determined by the difference in `true_rank` between opponents, using predefined rules.
- **CFP Points System:** Teams earn CFP points based on game results and the relative CFP rankings of their opponents.
- **Weekly Rankings:** CFP rankings are updated weekly based on cumulative CFP points, with ties broken by the previous week's order.

3.1.2 Rules for Game Outcomes

The probability of a team winning is calculated as follows:

- Teams with a **true rank difference of 5 or less** have a 50% chance of winning.
- For a **true rank difference of 6-10**, the better team wins 75% of the time.
- For a **true rank difference of 11-20**, the better team wins 90% of the time.
- For a **true rank difference greater than 20**, the better team wins 99% of the time.

3.1.3 CFP Points System Explanation

The CFP points system determines how many points teams earn based on the outcome of a game and the relative CFP rankings of the opponents. The system is designed to reward wins over stronger opponents and penalize losses to weaker ones. The following table outlines each situation and the corresponding points awarded:

Opponent's CFP Rank	Situation Description	CFP Points
Opponent's CFP rank is better than yours	Defeating a stronger opponent	5
Up to 5 spots lower than your rank	Beating a similarly ranked team	5
More than 5 spots lower than your rank	Beating a weaker opponent	4
Opponent's CFP rank is better than yours	Losing to a stronger opponent	3
1–4 spots lower than your rank	Losing to a similarly ranked team	2
5–14 spots lower than your rank	Losing to a weaker opponent	1
15 or more spots lower than your rank	Losing to a far weaker opponent	0

Table 1: CFP Points System: Breakdown by Game Outcome and Relative Rankings

Key Notes on the CFP Points System:

- **Winning Rewards:** Wins against stronger or similarly ranked opponents are valued highest (5 points). Wins against significantly weaker teams are devalued (4 points).
- **Losing Penalties:** Losses to stronger teams incur a moderate penalty (3 points), while losses to weaker teams result in fewer or no points, depending on the ranking gap.
- **Ranking Gap Influence:** The larger the gap between the team's rank and the opponent's rank, the more the outcome influences the CFP points awarded, reflecting the expected difficulty of the matchup.

This points system directly impacts how teams accumulate season points and move up or down in the weekly CFP rankings, creating a dynamic feedback loop that magnifies the importance of preseason rankings and game outcomes.

3.1.4 Visualization

The results of the simulation are visualized using a line graph that tracks each team's CFP ranking over the weeks. The graph highlights how the incorrect initial rankings create distortions that persist throughout the season.

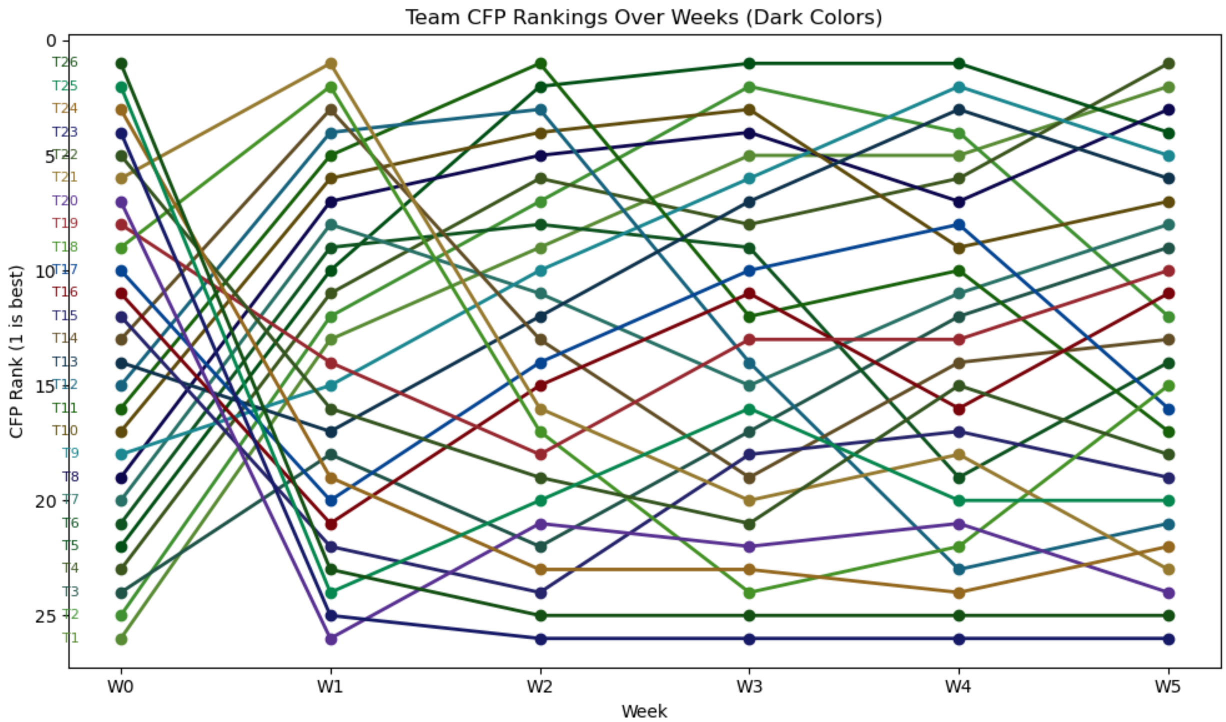

Figure 1: CFP Rankings Over Five Weeks with Inverse Initial Poll

3.1.5 A Harsher Committee Approach

In this scenario, we modify the CFP points system to reward winners more aggressively while penalizing losers more severely. This approach reflects a committee that values on-field outcomes over preconceived notions, emphasizing tangible performance rather than fluke results. The following table details the updated CFP points system:

Opponent's CFP Rank	Situation Description	CFP Points
Opponent's CFP rank is better than yours	Defeating a stronger opponent	8
Up to 5 spots lower than your rank	Beating a similarly ranked team	8
More than 5 spots lower than your rank	Beating a weaker opponent	6
Opponent's CFP rank is better than yours	Losing to a stronger opponent	2
1–4 spots lower than your rank	Losing to a similarly ranked team	1
5 or more spots lower than your rank	Losing to a weaker opponent	0

Table 2: Updated CFP Points System: Emphasizing Outcomes Over Perception

Qualitative Changes:

- **Higher Rewards for Wins:** Winning against a higher-ranked or similarly ranked team yields a significant 8 points, emphasizing on-field dominance.

- **Harsher Penalties for Losses:** Losses to much lower-ranked teams are punished severely, with zero points awarded.
- **Reduced Influence of Perception:** The system minimizes the impact of preseason biases by focusing on actual performance and neglecting statistical anomalies or luck.

The harsher system ensures that teams are judged primarily by their results, not their starting position in the preseason poll. This, essentially, leaves the results of a team up to their performance.

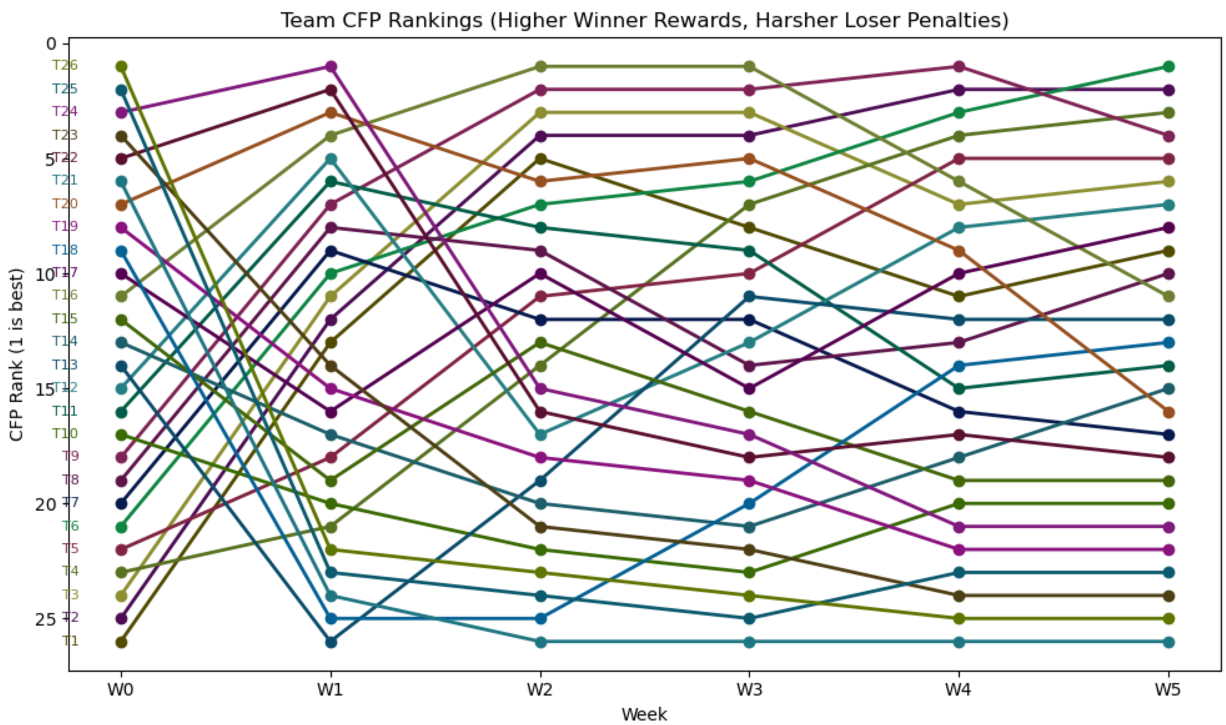

Figure 2: CFP Rankings Over Five Weeks with Harsher Committee Rules

3.1.6 Analysis

The graphs reveal distinct patterns between the standard and harsher committees, showcasing how different approaches affect rankings over the season.

Standard Committee: Under the standard CFP points system, the rankings evolved gradually, with teams slowly converging toward their true positions. Notably, middle-of-the-pack teams experienced minimal volatility, creating a more stable ranking environment. This gradual adjustment allowed for a relatively "correct" assessment of team strength by the end of the season, minimizing overreactions to individual game outcomes.

Harsher Committee: In contrast, the harsher committee produced more dramatic early-season shifts. Teams quickly rose or fell based on immediate results, reflecting the system's greater emphasis on rewarding wins and penalizing losses. This led to two significant qualitative effects:

- **Excitement in Early Season:** The rapid movement in rankings created dramatic rises and falls, enhancing the narrative for fans and media coverage. Teams could make meteoric rises only to crash following a single loss.
- **Stability in Late Season:** By the end of the season, most top teams had settled into their ultimate positions, leaving little suspense for late-season games. Only middle-ranked teams continued to experience volatility, creating a contrast to the early-season drama.

Comparison: While the harsher committee approach prioritized on-field results, it often exaggerated short-term trends, creating a roller-coaster effect in rankings. In contrast, the standard system's slower adjustments allowed for a more tempered evaluation, reducing the impact of individual anomalies and delivering a more balanced ranking over time.

3.2 The Inverse Ranking Scenario: Full FBS Model

3.2.1 Introduction to the Full FBS Model

The Full FBS Model expands the simulation from the smaller 26-team model to encompass all 134 FBS teams across a full 12-week season. While the smaller simulations were instrumental in understanding instantaneous effects of different committee styles (standard versus harsher), they were intentionally constrained for clarity and simplicity. The smaller model allowed for easier tracking of team trajectories, providing insights into how committee decisions affected rankings on a granular level. In contrast, the Full FBS Model aims to evaluate long-term trends and average outcomes by simulating the complete ecosystem of college football. Key differences include:

- **Full Dataset:** Includes all 134 FBS teams, providing a realistic representation of the college football hierarchy.
- **Extended Duration:** Simulates a full 12-week season to assess ranking evolution over time.
- **Multiple Runs:** Each simulation is repeated 100 times to generate average outcomes, minimizing the influence of random fluctuations.
- **Enhanced Metrics:** Tracks metrics such as average ranking difference (AvgDiff), maximum ranking difference (MaxDiff), biggest weekly rank rise (MaxRise), and biggest weekly rank fall (MaxFall).

This comprehensive model enables a more robust evaluation of committee decision-making and the dynamics of rankings throughout a season.

3.2.2 Game Logic

The probability of a team winning is calculated based on the difference in their true ranks, similar to the smaller model:

- For a **true rank difference of 5 or less**, the game is considered a toss-up, and each team has a 50% chance of winning.
- For a **true rank difference of 6-10**, the better team wins 65% of the time.
- For a **true rank difference of 11-15**, the better team wins 75% of the time.
- For a **true rank difference of 16-25**, the better team wins 85% of the time.
- For a **true rank difference of 26-50**, the better team wins 95% of the time.
- For a **true rank difference of 51-100**, the better team wins 98% of the time.
- For a **true rank difference greater than 100**, the better team wins 99% of the time.

This structured approach ensures that stronger teams generally prevail while still allowing for the unpredictability and excitement of occasional upsets. The same rules apply to both the smaller simulation model and the Full FBS Model, scaled to account for larger ranges in true rank differences.

3.2.3 Interpretation of Metrics:

To avoid relying on an eye test of hectic, full-FBS rankings graphs, it's important we collect valuable data as we simulate varying circumstances. This brings in a total of six new metrics to be tracked:

- **AvgDiff:** Measures the average absolute difference between CFP rankings and true rankings. A lower AvgDiff indicates better predictive accuracy, improving steadily over the season.
- **MaxDiff:** Represents the largest absolute discrepancy in rankings. The gradual decline in MaxDiff shows the committee's ability to resolve outliers over time.
- **MaxRise and MaxFall:** Indicate the largest ranking movements in a single week. High values in early weeks suggest greater excitement and unpredictability, which stabilize in later weeks as teams settle into their appropriate tiers.
- **AvgDiff25 and MaxDiff25:** Indicate the same as AvgDiff and MaxDiff, but solely for the committee's top 25 teams for that week. This is relevant specifically for playoff birth discussions, as imperfections anywhere past the 25th team normally is irrelevant for the CFP committee.

3.2.4 Standard Committee Approach

Under the standard committee model, rankings evolved methodically over the course of 12 weeks, reflecting the system's conservative adjustments. This approach, while slower, provided stability and minimized overreactions to single-game outcomes. Observations include:

- **Controlled Movements:** Teams adjusted positions more gradually, reducing erratic jumps in the rankings.
- **Improved Predictive Accuracy:** The final rankings closely aligned with the true team strengths, reflecting a well-balanced approach.
- **Excitement Trade-off:** The conservative nature of the standard committee resulted in fewer dramatic shifts towards the beginning of the season, which could reduce early-season excitement for fans.

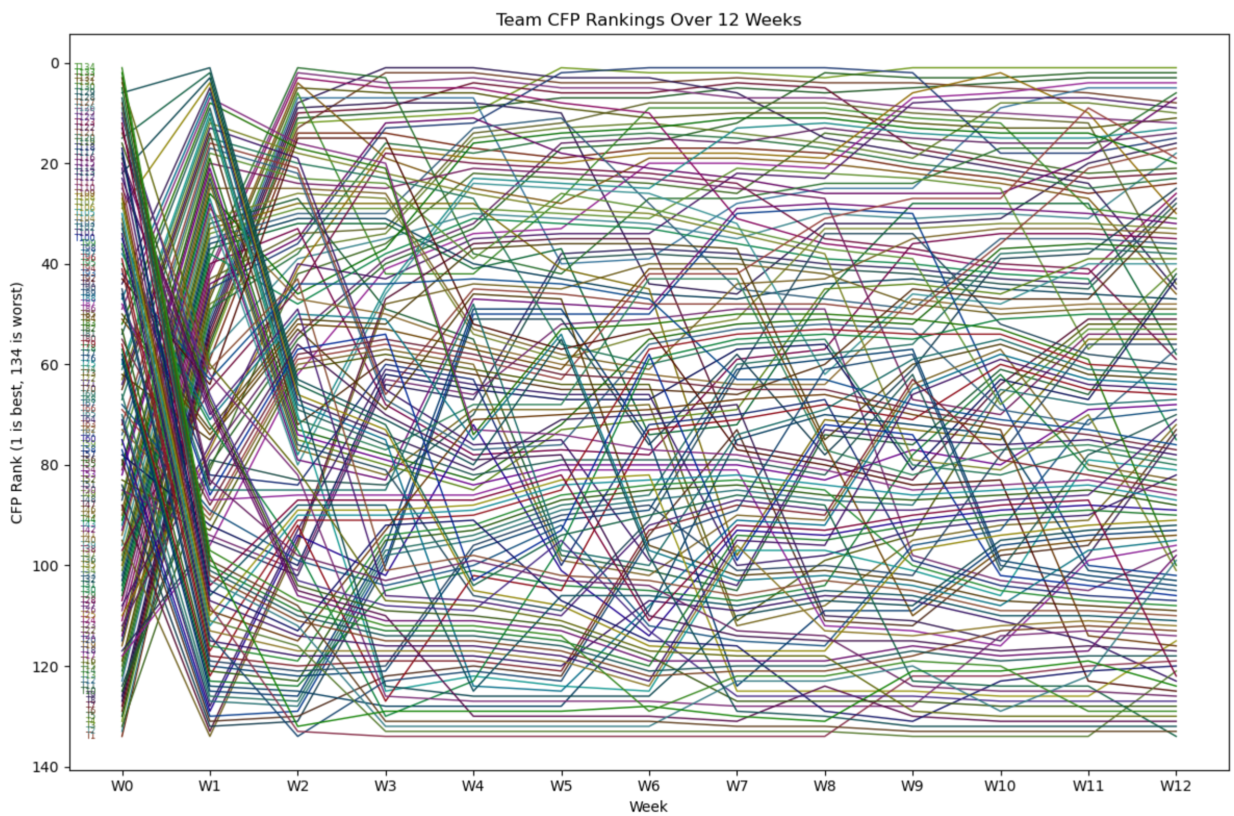

Figure 3: Team Rankings Over 12 Weeks (Standard Committee)

Metrics Analysis: The table below summarizes key metrics from the standard committee model, averaged over 100 runs. These metrics help assess both the accuracy and excitement generated by the committee's decisions.

Week	AvgDiff	MaxDiff	MaxRise	MaxFall	AvgDiff25	MaxDiff25
0	67.00	133.00	0.00	0.00	109.00	133.00
1	37.12	118.89	71.81	90.24	63.27	118.88
2	26.28	90.90	48.07	64.74	31.49	87.55
3	21.69	80.10	37.49	54.61	21.25	70.92
4	19.41	73.32	34.75	50.35	16.84	59.96
5	18.07	69.49	32.80	48.97	14.57	52.11
6	17.12	66.45	30.62	45.89	13.58	47.71
7	16.46	64.58	29.89	45.15	13.05	46.40
8	16.09	62.75	30.00	44.43	12.46	44.18
9	15.67	61.88	28.72	42.96	11.78	41.57
10	15.40	61.59	28.28	42.86	11.68	40.40
11	15.12	58.76	28.12	41.55	11.67	39.54
12	14.89	58.88	27.45	40.63	11.50	39.02

Table 3: Weekly Averages Over 100 Runs: Standard Committee Model

These results highlight the trade-offs inherent in the standard committee approach, balancing accuracy with the need for dynamic, engaging rankings. These results are provided in graph format under Appendix A.

3.2.5 Harsher Committee Approach

The harsher committee approach modifies the CFP scoring system to reward winning teams more aggressively and penalize losing teams more severely. While these scoring values are arbitrary, their use is justified within this simulation framework as the aim is to evaluate holistic trends rather than specific real-world accuracy. Comparing overall outcomes between the standard and harsher committees provides insights into how changes in scoring systems influence rankings and, by extension, the narrative of a college football season.

The harsher scoring system reflects the following principles:

- **Greater Emphasis on Results:** Teams that win against higher-ranked opponents or comparable competition are rewarded more significantly, creating dramatic shifts in rankings.

- **Holistic Comparison:** The absolute values of CFP points are less relevant than the trends observed across multiple runs, allowing for meaningful comparisons between models.

- **Excitement Factor:** The increased volatility in rankings introduces unpredictability, which can appeal to fan engagement and media coverage.

Changes in Scoring System: The harsher scoring system differs from the standard committee in the following ways:

- Wins against better-ranked or similarly ranked opponents are awarded significantly higher points.
- Losses against much lower-ranked teams result in sharper penalties, amplifying the effect of perceived upsets.
- The system prioritizes on-field outcomes, disregarding pre-existing perceptions or small fluctuations in team strength.

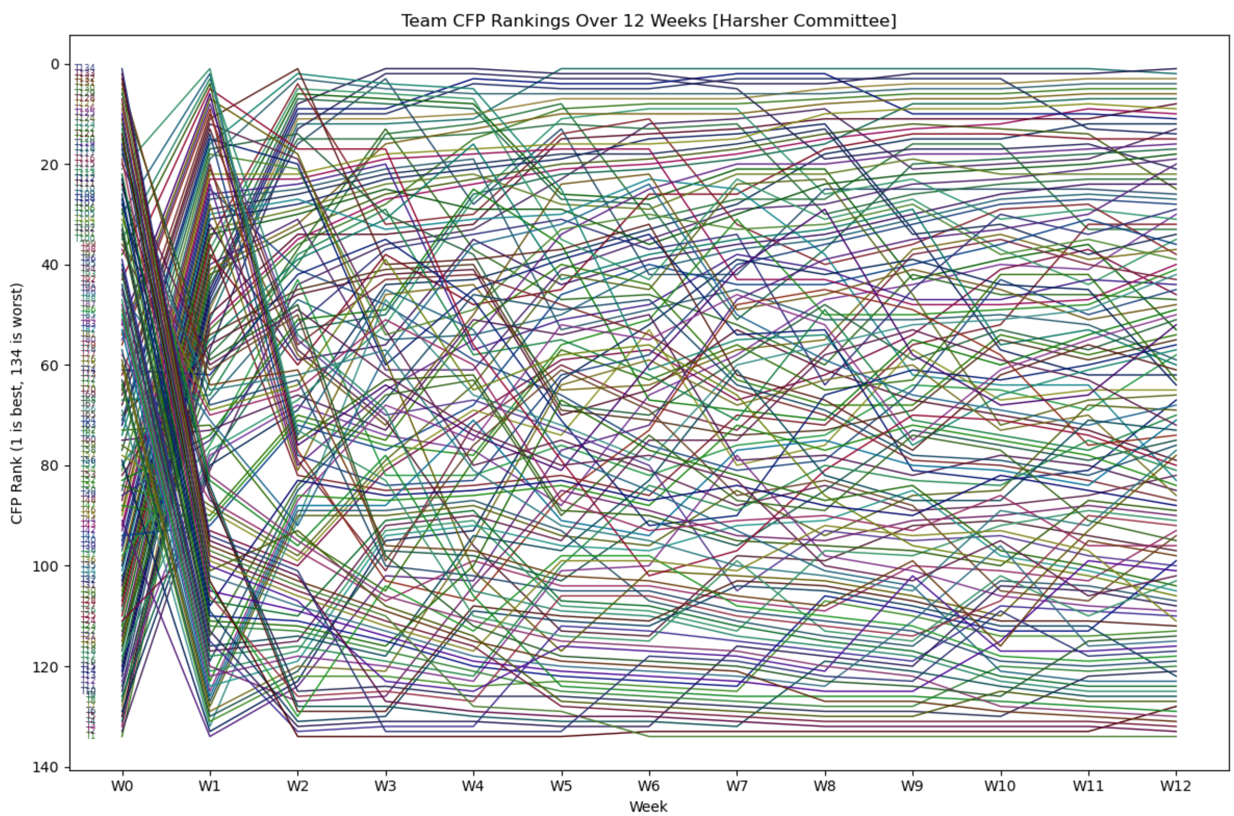

Figure 4: Team Rankings Over 12 Weeks (Harsher Committee)

Week	AvgDiff	MaxDiff	MaxRise	MaxFall	AvgD25	MaxD25
0	67.00	133.00	0.00	0.00	109.00	133.00
1	36.48	114.65	74.75	90.39	60.16	114.63
2	26.25	91.95	50.03	61.89	31.08	87.70
3	21.57	80.21	39.75	45.67	20.78	71.49
4	18.86	71.96	32.96	39.38	16.13	58.46
5	17.21	65.97	28.92	34.72	13.91	49.91
6	15.86	62.43	26.69	31.87	12.83	46.38
7	14.93	59.03	24.24	29.08	11.91	42.49
8	14.12	55.91	22.74	27.13	11.26	39.87
9	13.44	53.10	21.31	25.27	10.77	38.23
10	12.90	50.61	20.34	23.81	10.26	35.73
11	12.41	48.11	19.39	22.78	9.87	32.92
12	12.02	46.15	18.67	21.67	9.59	32.58

Table 4: Weekly Averages Over 100 Runs: Harsher Committee Model

These results are provided in graph format under Appendix B.

3.2.6 Analysis of Results

The harsher committee produced distinct patterns and outcomes, highlighting its impact on rankings:

AvgDiff and MaxDiff:

- **AvgDiff:** Both the standard and harsher committees exhibited a steady decline in AvgDiff over time. By Week 12, the AvgDiff for the harsher committee reached 12.12, compared to 15.01 for the standard committee. This suggests that the harsher committee converged rankings toward true team strengths more efficiently, albeit with higher early-season volatility.
- **MaxDiff:** The maximum discrepancies for both models decreased substantially. The harsher committee reduced MaxDiff from 133.00 in Week 0 to 46.05 in Week 12, while the standard committee achieved a MaxDiff of 58.76 by Week 12. This demonstrates the harsher committee's stronger emphasis on immediate results, resolving large discrepancies more aggressively.
- **Implications:** The harsher committee's quicker convergence supports its utility in achieving accuracy earlier in the season, though it may come at the cost of greater initial instability.

MaxRise and MaxFall:

- **MaxRise:** The largest weekly rank improvements peaked early for both models, with the harsher committee reaching 75.09 in Week 1 compared to 71.77 for the standard committee. By Week 12, MaxRise values were 18.45 and 27.45, respectively, reflecting a more dramatic stabilization in the harsher model.
- **MaxFall:** Similarly, the largest rank drops began at 90.65 for the harsher committee and 90.24 for the standard, decreasing to 21.47 and 40.52, respectively, by Week 12. The harsher committee's sharper declines highlight its greater volatility in handling outliers.
- **Implications:** Early volatility in the harsher committee contributes to excitement and unpredictability, but it also risks undermining perceived fairness if dramatic rises and falls are not consistently justified.

Critiques and Considerations: While the harsher committee demonstrated certain advantages, several critiques must be addressed:

- **Unrealistic Initial Conditions:** The inverse preseason ranking is a deliberately extreme scenario. While useful for testing, it is unlikely to reflect real-world preseason rankings.
- **Randomized Schedules:** Conference schedules in the simulation are randomized, unlike real-world scenarios where schedules are determined by rivalries and traditions. This is something to be addressed in a later section.
- **Lack of In-Game Performance Metrics:** Close wins and blowouts are treated equally, which may distort rankings. Repeated simulations mitigate this effect but do not eliminate it entirely.
- **Arbitrary Game Logic:** The probability model for game outcomes is not based on historical data, potentially skewing the simulation toward being upset-heavy or upset-weak in certain runs.

3.2.7 Qualitative Takeaways

The comparative analysis of the harsher and standard committee models reveals significant differences in their outcomes, implications for rankings, and qualitative impacts on college football.

Accuracy and Convergence:

- **Standard Committee:** The gradual adjustments under the standard committee provided a measured convergence to accurate rankings, reaching an AvgDiff of 15.01 by Week 12. This approach minimizes drastic movements and overreactions, yielding a stable and reliable system for reflecting true team strengths over time.

- **Harsher Committee:** The harsher committee achieved greater accuracy more quickly, with an AvgDiff of 12.12 by Week 12, highlighting its efficiency in correcting preseason biases and aligning rankings with on-field performance. However, the early-season volatility may introduce short-term inaccuracies as teams rapidly rise or fall.

Excitement and Fan Engagement:

- **Standard Committee:** The slower pace of changes results in a more predictable ranking system, which may appeal to purists valuing tradition and reliability. However, it can reduce early-season excitement as rankings tend to stabilize quickly, leaving fewer surprises.
- **Harsher Committee:** Watching a team dramatically rise through the rankings after key wins under the harsher committee could lead to more engaging narratives and better television ratings. The rapid inclusion of previously underestimated teams creates opportunities for unexpected storylines, energizing fan bases and driving media interest. However, this same volatility may lead to frustrations as teams fall equally quickly, potentially diminishing confidence in the system.

Impacts on Team Strategy:

- **Standard Committee:** Teams may focus on consistency, knowing that the system rewards gradual improvement and penalizes large fluctuations less harshly. This could encourage disciplined play over dramatic risks.
- **Harsher Committee:** The harsher model's emphasis on immediate results could incentivize riskier strategies, such as scheduling tougher opponents earlier in the season to maximize rankings or adopting aggressive playstyles to ensure decisive victories.

Media and Narrative Building:

- **Standard Committee:** The standard committee provides a more consistent foundation for evaluating teams, maintaining the integrity of traditional ranking hierarchies and reducing sensationalism.
- **Harsher Committee:** The harsher committee's greater volatility fosters dramatic narratives, making it easier to build excitement around surprising upsets or underdog success stories.

4 Takeaways and Future Directions

The findings from this study provide a comprehensive evaluation of how different committee decision-making approaches impact College Football Playoff (CFP) rankings. By examining both the standard and harsher committee models, several data-driven and qualitative takeaways emerge, alongside important considerations for improving future simulations.

4.1 Key Takeaways

- **Harsher Committee Accuracy:** The harsher committee consistently produced rankings that more closely aligned with true team strengths. By Week 12, metrics such as `AvgDiff` and `MaxDiff` showed greater convergence under the harsher approach compared to the standard committee, highlighting its effectiveness in rewarding on-field performance.

- **Excitement vs. Stability:** The harsher committee generated significant early-season excitement, with larger `MaxRise` and `MaxFall` values as teams experienced dramatic ranking shifts. However, this volatility could be unsettling for traditionalists, as strong teams risked exclusion from playoff contention due to isolated losses.

- **Predictive Stability in Standard Committee:** While less responsive to game outcomes, the standard committee approach offered greater stability and predictability. This approach minimized ranking volatility and emphasized consistent performance over the course of the season.

- **Impact on Fan Engagement:** The harsher approach's dynamic rankings may appeal to broader audiences, driving narratives of underdog rises and dramatic falls, while the standard committee's conservative adjustments provided a more predictable and measured progression.

4.2 Concerns and Limitations

Despite these insights, the study has several limitations that must be acknowledged:

- **Simplistic Game Logic:** The game outcome probabilities were based on predefined rank differences rather than real-world data. Incorporating historical performance metrics, such as team efficiency or in-game statistics, could improve simulation accuracy.

- **Unrealistic Initial Conditions:** The inverse ranking scenario was deliberately extreme and unlikely to reflect actual preseason rankings. While useful for testing systemic biases, future simulations should incorporate more nuanced initial conditions, such as weighted rankings or partial randomness.

- **Lack of Conference Dynamics:** Randomized schedules in the simulation do not account for conference-specific rivalries or uneven scheduling practices, which play a significant role in real-world CFP decisions.

- **Equal Weighting of Wins and Losses:** The current model treats all wins and losses equally, ignoring the context of game outcomes, such as margin of victory or the strength of opponent performance.

4.3 Future Directions

To address these limitations and enhance the robustness of the system, several improvements are proposed:

- **Tuned Game Logic:** Develop a more sophisticated game logic framework that incorporates team efficiency metrics, strength of schedule, and other advanced analytics to simulate outcomes more realistically.
- **Improved Initial Rankings:** Replace the inverse ranking model with a tiered or weighted ranking system that reflects real-world preseason expectations while introducing controlled variability to simulate perception biases.
- **Contextual Scoring Adjustments:** Refine the CFP points system to include contextual factors such as margin of victory, game location (home vs. away), and injury impacts, aligning more closely with committee decision-making practices.
- **Integration of Conference Dynamics:** Implement scheduling constraints that reflect real-world conference structures, rivalries, and uneven competition levels, providing a more accurate representation of the college football ecosystem.
- **Long-Term Performance Evaluation:** Extend simulations to include multiple seasons, enabling an analysis of cumulative effects and trends in ranking methodologies.

4.4 Concluding Remarks

This study underscores the trade-offs inherent in CFP ranking methodologies, particularly between accuracy, excitement, and stability. While the harsher committee demonstrated superior alignment with true team strengths, its volatility raises concerns about fairness and consistency. Addressing the outlined limitations and integrating more nuanced features into future models will provide deeper insights into the dynamics of college football rankings and improve the system's ability to fairly evaluate team performance.

Acknowledgments

This paper was inspired by Connor Stallions' analysis of Strength of Schedule and its limitations. His insights provided a valuable foundation for this discussion.

A Figures for Standard, Inverse Committee Analysis

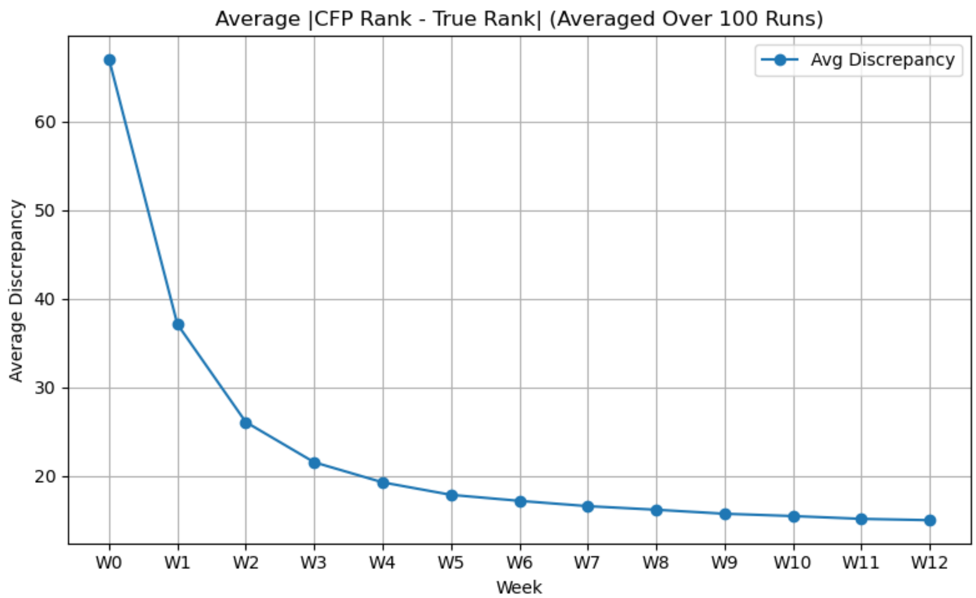

Figure 5: Average Ranking Difference Over Time

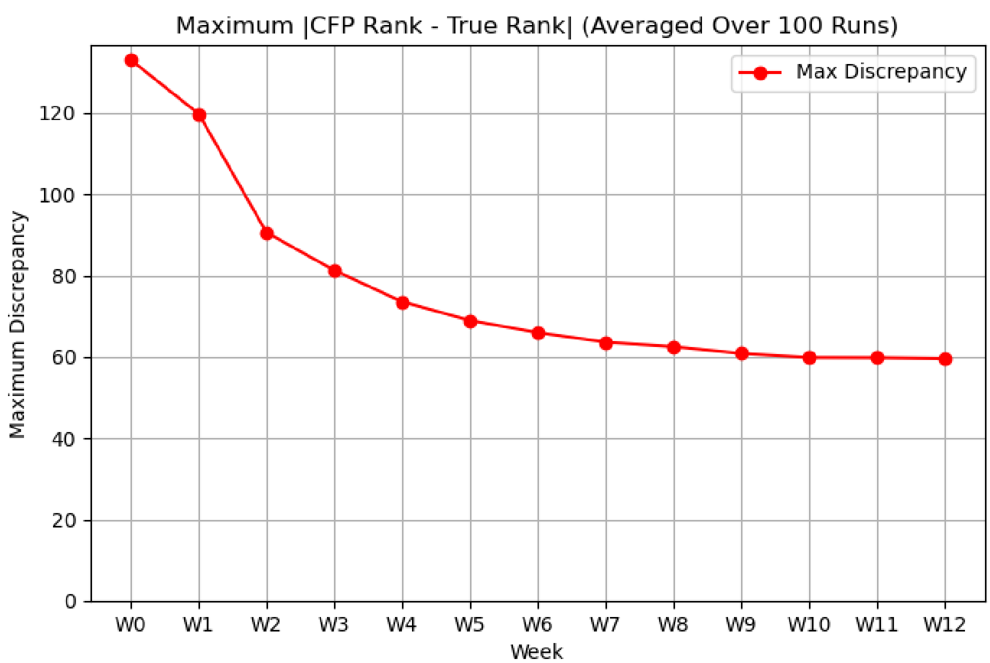

Figure 6: Maximum Ranking Difference Over Time

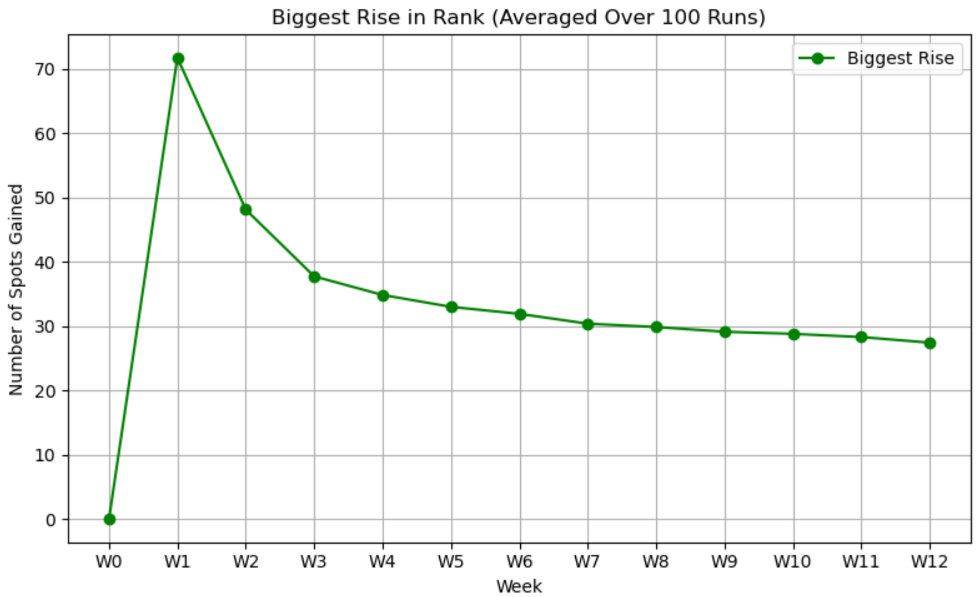

Figure 7: Biggest Weekly Rank Rise Over Time

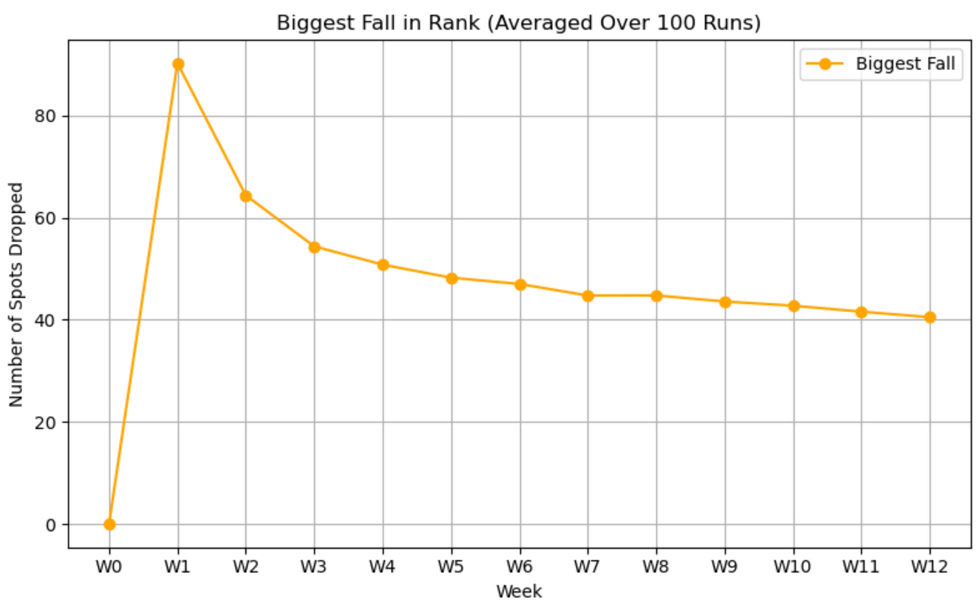

Figure 8: Biggest Weekly Rank Fall Over Time

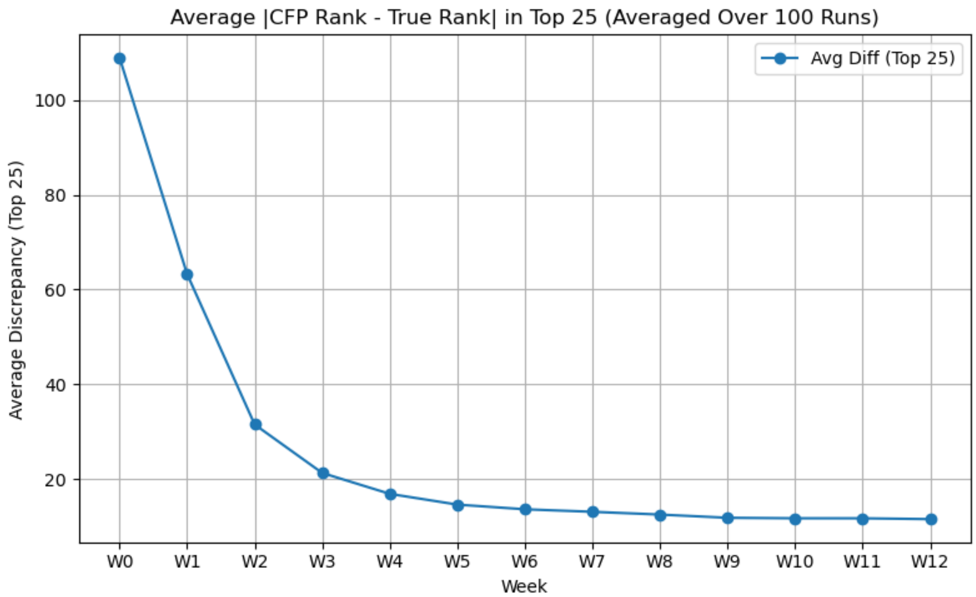

Figure 9: Average Ranking Difference Over Time Within Top-25

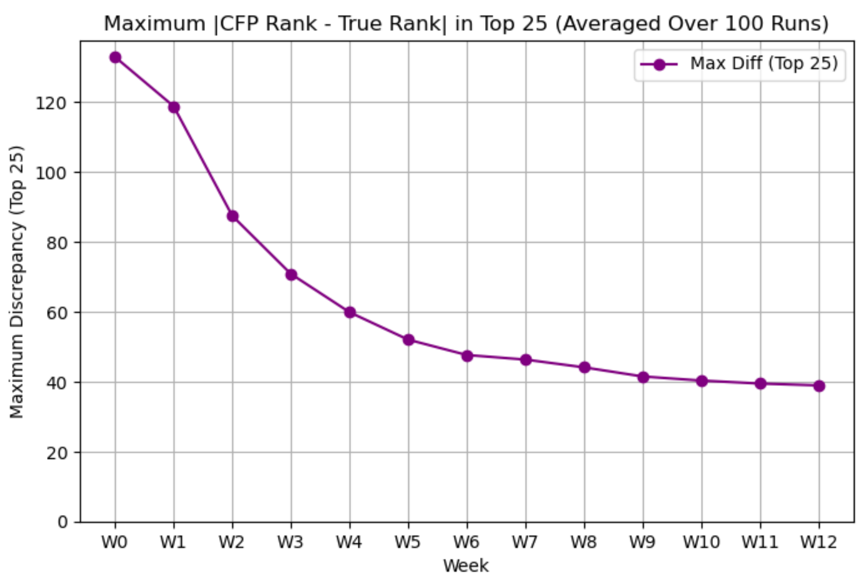

Figure 10: Maximum Ranking Difference Over Time Within Top-25

B Figures for Harsher, Inverse Committee Analysis

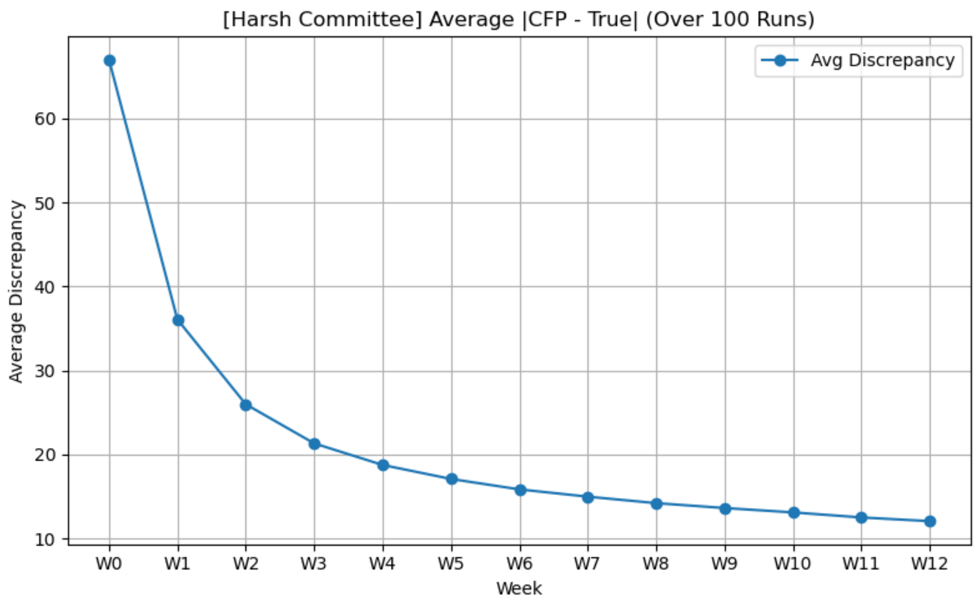

Figure 11: Average Ranking Difference Over Time (Harsher Committee)

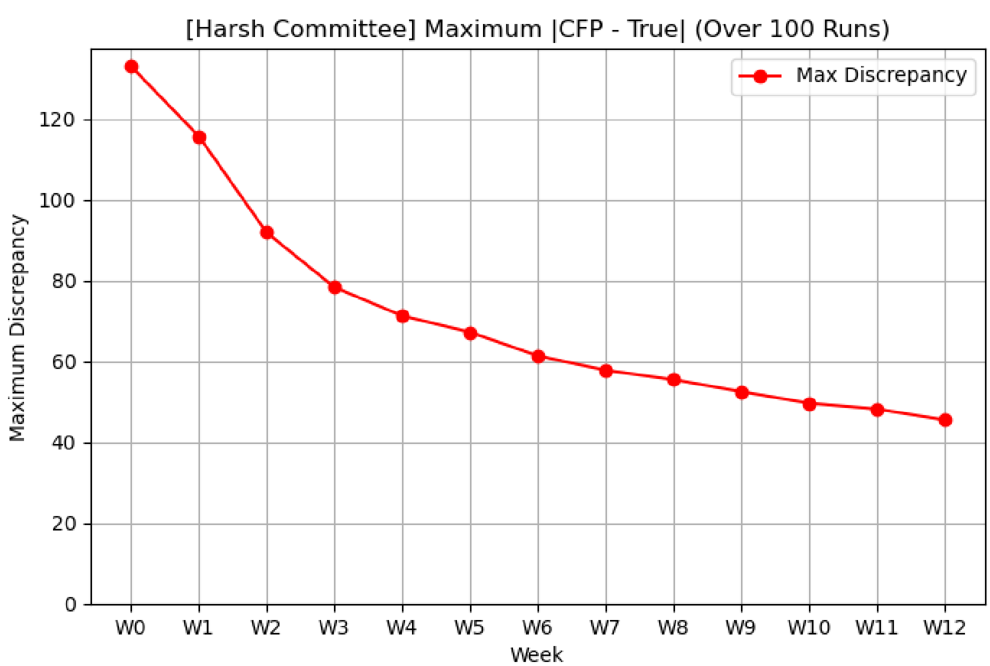

Figure 12: Maximum Ranking Difference Over Time (Harsher Committee)

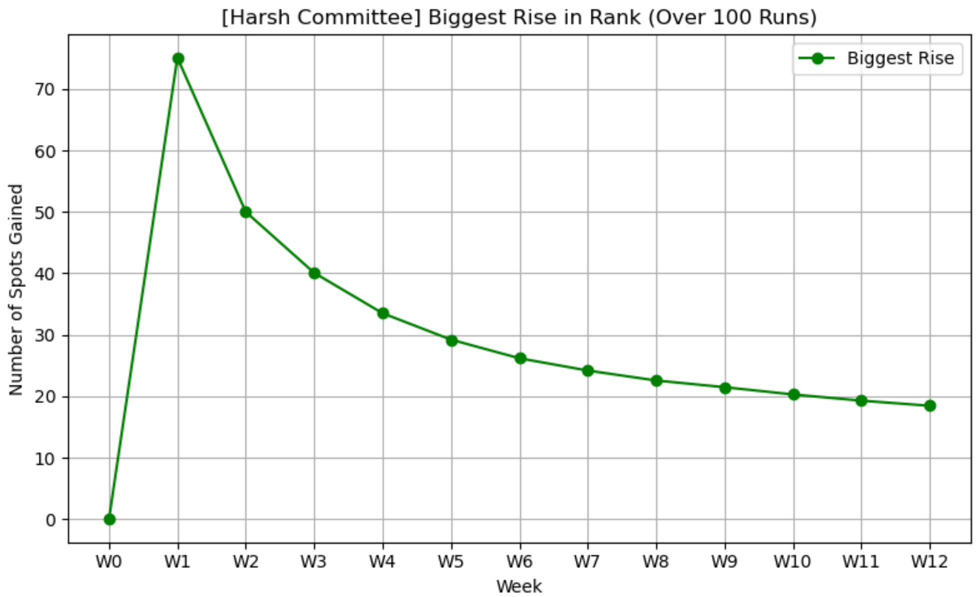

Figure 13: Maximum Weekly Rank Rises Over Time (Harsher Committee)

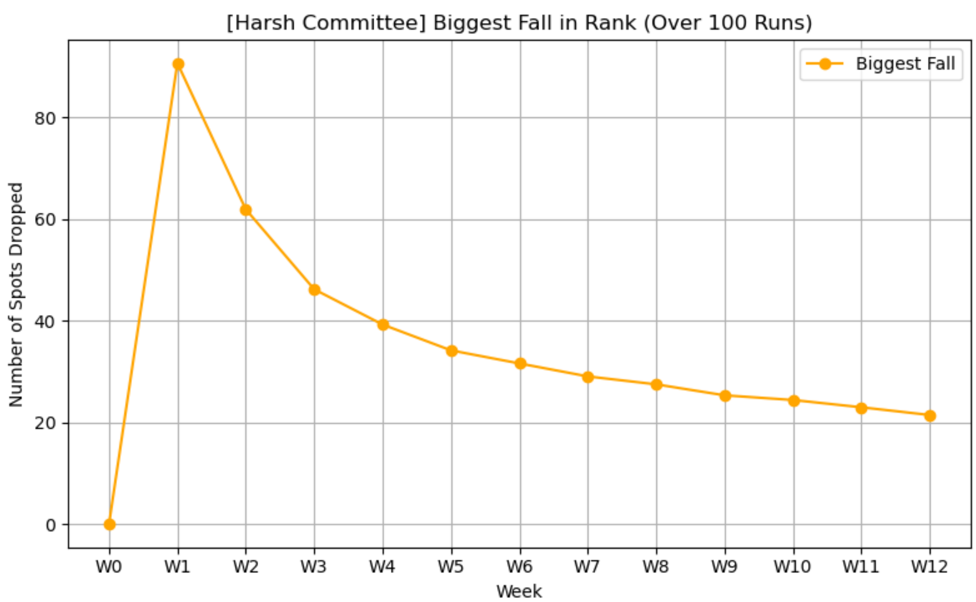

Figure 14: Maximum Weekly Rank Drops Over Time (Harsher Committee)

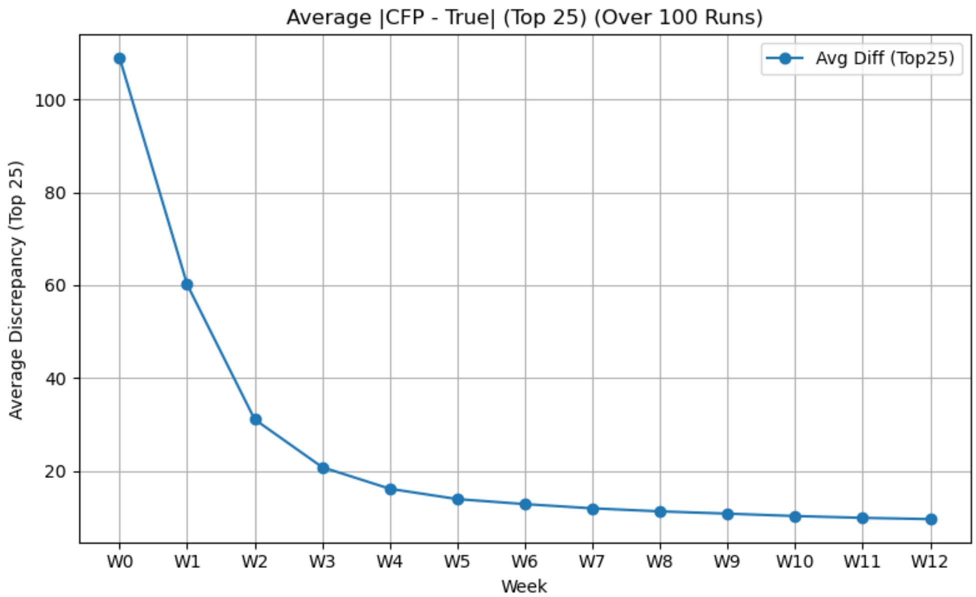

Figure 15: Average Ranking Difference Over Time Within Top-25

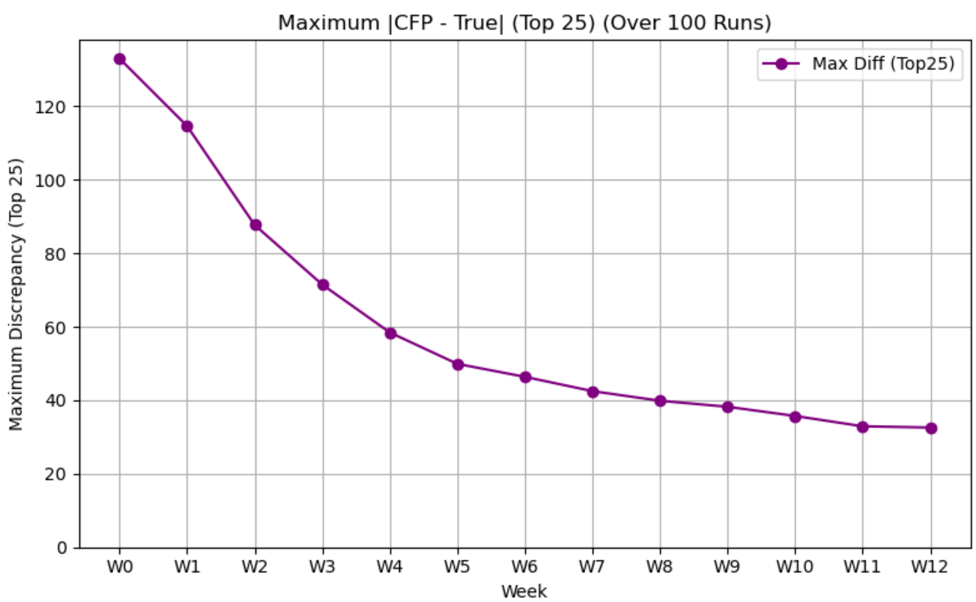

Figure 16: Maximum Ranking Difference Over Time Within Top-25

C Python Codes

C.1 Standard Committee Small Inverse Model

```python
import random
import copy
import matplotlib.pyplot as plt
import colorsys  # for generating dark colors that are easy to read on the graph

def random_dark_color():
    """
    Generate a random 'dark-ish' color in RGB with no alpha,
    by picking a random hue, fairly high saturation, and lower brightness.
    """
    h = random.random()              # 0..1
    s = random.uniform(0.6, 1.0) # fairly high saturation
    v = random.uniform(0.3, 0.6) # keep brightness below ~60%

    r, g, b = colorsys.hsv_to_rgb(h, s, v)
    return (r, g, b)

def generate_teams(num_teams=26):
    """
    Generate a list of team dictionaries.
    Each has:
      - name (e.g. "Team #1" ... "Team #26")
      - true_rank (1 to 26, 1 is best)
      - cfp_rank (inverse: #1 -> cfp_rank=26, #26-> cfp_rank=1)
      - season_points (tracks CFP points across the season, starts at 0)
    """
    teams = []
    for i in range(1, num_teams + 1):
        true_rank = i
        # Inverse CFP rank: best team (#1) -> 26, worst (#26) -> 1
        cfp_rank = num_teams - i + 1
        team_dict = {
            'name': f"Team #{i}",
            'true_rank': true_rank,
            'cfp_rank': cfp_rank,
            'season_points': 0
```

```python
        }
        teams.append(team_dict)
    return teams

def probability_of_win(team_a_true, team_b_true):
    """
    Return probability that team_a (lower true_rank = better) beats team_b.
    Probability rules (based on absolute difference):
      -     5    => 50/50
      -   10     => better team 75%
      -   20     => better team 90%
      - >20   => better team 99%
    """
    diff = team_b_true - team_a_true  # positive => B is worse
    abs_diff = abs(diff)

    if abs_diff <= 5:
        return 0.5
    elif abs_diff <= 10:
        return 0.75 if diff > 0 else 0.25
    elif abs_diff <= 20:
        return 0.90 if diff > 0 else 0.10
    else:
        return 0.99 if diff > 0 else 0.01

def determine_cfp_points(team_cfp_rank, opponent_cfp_rank, did_win):
    """
    Award CFP points based on last week's ranks and the game result:
      - 0  losing to a team   15   spots behind you in the ranking
      - 1  losing to a team 5  14   spots behind you
      - 2  losing to a team 1  4   spots behind you
      - 3  losing to a stronger team (opponents rank < yours)
      - 4  beating a team more than 5 spots below you
      - 5  beating a stronger team or one up to 5 spots below you
    """
    if did_win:
        # Win:
        if (opponent_cfp_rank < team_cfp_rank) or ((opponent_cfp_rank - team_cfp_rank) <= 5):
            return 5
```

```
        else:
            return 4
    else:
        # Loss:
        if opponent_cfp_rank < team_cfp_rank:
            return 3
        diff = opponent_cfp_rank - team_cfp_rank
        if diff >= 15:
            return 0
        elif diff >= 5:
            return 1
        elif diff >= 1:
            return 2
        return 2  # fallback, in case my game logic isn't airtight

def break_ties(teams_sorted, teams_last_week):
    """
    Reorder ties so that if two teams have the same season_points,
    they remain in the same relative order as last week's standings.
    This is something we typically see out of the CFP Committee.
    """
    name_to_idx_last_week = {t['name']: i for i, t in enumerate(teams_last_week)}

    # stable 2-pass sort:
    #  1) ascending by last week's index
    #  2) descending by season_points
    teams_sorted.sort(key=lambda t: name_to_idx_last_week[t['name']])
    teams_sorted.sort(key=lambda t: t['season_points'], reverse=True)
    return teams_sorted

def simulate_season(num_teams=26, num_weeks=5, seed=None):
    """
    Simulate a season for 26 teams, 5 weeks by default.
    Returns a list of weekly snapshots (including preseason=week0).
    """
    if seed is not None:
        random.seed(seed)
    else:
```

```
        random.seed()

    teams = generate_teams(num_teams)

    # Preseason ordering by cfp_rank
    current_cfp_order = sorted(teams, key=lambda t: t['cfp_rank'])
    # Storing a deep-copied snapshot each week
    weekly_rankings = [copy.deepcopy(current_cfp_order)]

    for week in range(1, num_weeks + 1):
        indices = list(range(num_teams))
        random.shuffle(indices)

        # Pair them up
        matchups = []
        for i in range(0, num_teams, 2):
            matchups.append((indices[i], indices[i+1]))

        # Look up last week's cfp rank (1-based)
        last_week_map = {t['name']: idx+1 for idx, t in enumerate(current_cfp_order)}

        # Simulate games
        for idx_a, idx_b in matchups:
            team_a = teams[idx_a]
            team_b = teams[idx_b]
            p_a_wins = probability_of_win(team_a['true_rank'], team_b['true_rank'])
            a_wins = (random.random() < p_a_wins)

            cfp_a = last_week_map[team_a['name']]
            cfp_b = last_week_map[team_b['name']]

            pts_a = determine_cfp_points(cfp_a, cfp_b, a_wins)
            pts_b = determine_cfp_points(cfp_b, cfp_a, not a_wins)

            team_a['season_points'] += pts_a
            team_b['season_points'] += pts_b

        # Re-sort by season_points desc, tie-break with last week's order
        teams_sorted = sorted(teams, key=lambda t: t['season_points'], reverse=True)
```

```
        new_cfp_order = break_ties(teams_sorted, current_cfp_order
    )

        # Update .cfp_rank in the new order
        for rank_pos, t_dict in enumerate(new_cfp_order):
            t_dict['cfp_rank'] = rank_pos + 1

        # Save deep copy snapshot for this week's final standings
        weekly_rankings.append(copy.deepcopy(new_cfp_order))

        # Next iteration
        current_cfp_order = new_cfp_order

    return weekly_rankings

def plot_rankings_over_weeks(weekly_rankings):
    """
    Plot each team's CFP rank across weeks (0..num_weeks),
    using only dark-ish random colors.
    Label each line on the left with that team's True Rank.
    """
    num_teams = len(weekly_rankings[0])

    # Building a dict: name -> [week0_rank, week1_rank, ...]
    name_to_ranks = {}
    name_to_true = {}

    for w, ranking_list in enumerate(weekly_rankings):
        for team in ranking_list:
            nm = team['name']
            if nm not in name_to_ranks:
                name_to_ranks[nm] = []
                name_to_true[nm] = team['true_rank']
            name_to_ranks[nm].append(team['cfp_rank'])

    # Generate dark colors for each team
    colors = [random_dark_color() for _ in range(num_teams)]

    fig, ax = plt.subplots(figsize=(10, 6))
    weeks_x = range(len(weekly_rankings))  # e.g. 0..5

    # Iterating in a stable order (e.g. sorted by true rank)
```

```
    all_names_sorted = sorted(name_to_ranks.keys(), key=lambda n: name_to_true[n])

    for idx, nm in enumerate(all_names_sorted):
        c = colors[idx % len(colors)]
        ranks_list = name_to_ranks[nm]
        true_r = name_to_true[nm]

        ax.plot(
            weeks_x,
            ranks_list,
            marker='o',
            color=c,
            linewidth=2
        )

        # Label on left at week0
        ax.text(
            -0.2,
            ranks_list[0],
            f"T{true_r}",
            ha='right',
            va='center',
            color=c,
            fontsize=8
        )

    # Flip y-axis so #1 is at top
    ax.invert_yaxis()
    ax.set_xlabel("Week")
    ax.set_ylabel("CFP Rank (1 is best)")
    ax.set_title("Team CFP Rankings Over Weeks (Dark Colors)")
    ax.set_xticks(list(weeks_x))
    ax.set_xticklabels([f"W{w}" for w in weeks_x])
    plt.tight_layout()
    plt.show()

def main():
    # 26 teams, 5 weeks
    weekly_data = simulate_season(num_teams=26, num_weeks=5, seed=None)

    # Print results
```

```python
    for w_idx, ranking_list in enumerate(weekly_data):
        if w_idx == 0:
            print("=== Preseason (Week 0) CFP Rankings ===")
        else:
            print(f"=== Week {w_idx} CFP Rankings ===")
        for t in ranking_list:
            print(f"CFP#{t['cfp_rank']:02d} - {t['name']} "
                  f"(TrueRank={t['true_rank']}) "
                  f"- Points={t['season_points']}")
        print()

    # Plot
    plot_rankings_over_weeks(weekly_data)

if __name__ == "__main__":
    main()
```

C.2 Harsh Committee Small Inverse Model

```python
import random
import copy
import matplotlib.pyplot as plt
import colorsys

def random_dark_color():
    """
    Generate a random dark-ish color in RGB
    with decent saturation and lower brightness,
    so it's easier to see on a white background.
    """
    h = random.random()
    s = random.uniform(0.6, 1.0)
    v = random.uniform(0.3, 0.6)
    r, g, b = colorsys.hsv_to_rgb(h, s, v)
    return (r, g, b)

def generate_teams(num_teams=26):
    """
    Create a list of team dictionaries.
    Each dictionary includes:
      - name (e.g. "Team #1", ... "Team #26")
      - true_rank (1..26, with 1 being the best)
      - cfp_rank (inverse preseason: best team #1 => cfp_rank=26, worst =>1)
      - season_points (tracks CFP points)
    We invert the preseason CFP so that a true_rank=1 gets cfp_rank=26, etc.
    """
    teams = []
    for i in range(1, num_teams + 1):
        true_rank = i
        # Inverse assignment for initial CFP
        cfp_rank = num_teams - i + 1
        teams.append({
            'name': f"Team #{i}",
            'true_rank': true_rank,
            'cfp_rank': cfp_rank,
            'season_points': 0
        })
    return teams
```

```python

def probability_of_win(team_a_true, team_b_true):
    """
    Given two teams' true ranks, return the probability team_a wins.
    Lower true_rank => stronger team.

    Probability rules (based on abs_diff):
      -       5    => 50/50
      -    10      => 75% (if team_a is stronger), else 25%
      -    20      => 90% vs. 10%
      - >20    => 99% vs. 1%
    """
    diff = team_b_true - team_a_true  # positive => team_a is better
    abs_diff = abs(diff)

    if abs_diff <= 5:
        return 0.5
    elif abs_diff <= 10:
        return 0.75 if diff > 0 else 0.25
    elif abs_diff <= 20:
        return 0.90 if diff > 0 else 0.10
    else:
        return 0.99 if diff > 0 else 0.01

# ------------------------------------------------------------
# UPDATED POINT SYSTEM (CFP)
# ------------------------------------------------------------
def determine_cfp_points(team_cfp_rank, opponent_cfp_rank, did_win):
    """
    New, more extreme system for CFP points:

    Winners:
      - 8 points: beat a stronger team (opponent cfp_rank < yours)
                  OR a team within 5 spots below
      - 6 points: beat a team more than 5 spots below

    Losers:
      - 2 points: lost to a stronger team (opponent cfp_rank < yours)
      - 1 point : lost to a team 1 4  cfp_rank spots behind you
```

```
      - 0 points: lost to a team  5  cfp_rank spots behind you (a 'bad loss')
    """
    diff = opponent_cfp_rank - team_cfp_rank
    if did_win:
        # Win
        if (opponent_cfp_rank < team_cfp_rank) or (diff <= 5):
            return 8
        else:
            return 6
    else:
        # Loss
        if opponent_cfp_rank < team_cfp_rank:
            # Lost to a stronger team => smaller penalty
            return 2
        else:
            if diff >= 5:
                return 0
            else:
                # diff in [1..4]
                return 1

def break_ties(teams_sorted, teams_last_week):
    """
    Sort primarily by season_points descending; if tie, keep last week's CFP order.
    """
    name_to_idx_last_week = {t['name']: i for i, t in enumerate(teams_last_week)}
    # stable two-pass sort
    teams_sorted.sort(key=lambda t: name_to_idx_last_week[t['name']])
    teams_sorted.sort(key=lambda t: t['season_points'], reverse=True)
    return teams_sorted

def simulate_season(num_teams=26, num_weeks=5, seed=None):
    """
    Simulate a multi-week season for 'num_teams' teams, awarding CFP points
    per the new (harsher) scoring rules.

    Returns a list of weekly_snapshots, including the preseason (
```

```
    week0).
    """
    if seed is not None:
        random.seed(seed)
    else:
        random.seed()

    teams = generate_teams(num_teams)

    # Preseason CFP order
    current_cfp_order = sorted(teams, key=lambda t: t['cfp_rank'])
    weekly_rankings = [copy.deepcopy(current_cfp_order)]

    for week in range(1, num_weeks + 1):
        indices = list(range(num_teams))
        random.shuffle(indices)
        matchups = [(indices[i], indices[i+1]) for i in range(0, num_teams, 2)]

        # Map from name->last week's cfp rank
        last_week_map = {
            t['name']: idx + 1 for idx, t in enumerate(current_cfp_order)
        }

        # Simulate each matchup
        for idx_a, idx_b in matchups:
            team_a = teams[idx_a]
            team_b = teams[idx_b]

            p_a_wins = probability_of_win(team_a['true_rank'], team_b['true_rank'])
            a_wins = (random.random() < p_a_wins)

            cfp_a = last_week_map[team_a['name']]
            cfp_b = last_week_map[team_b['name']]

            pts_a = determine_cfp_points(cfp_a, cfp_b, a_wins)
            pts_b = determine_cfp_points(cfp_b, cfp_a, not a_wins)

            team_a['season_points'] += pts_a
            team_b['season_points'] += pts_b

```

```
        # Re-sort by total CFP points
        teams_sorted = sorted(teams, key=lambda t: t['season_points'], reverse=True)
        new_cfp_order = break_ties(teams_sorted, current_cfp_order)

        # Update cfp_rank
        for rank_pos, t_dict in enumerate(new_cfp_order):
            t_dict['cfp_rank'] = rank_pos + 1

        weekly_rankings.append(copy.deepcopy(new_cfp_order))
        current_cfp_order = new_cfp_order

    return weekly_rankings

def plot_rankings_over_weeks(weekly_rankings):
    """
    Plot each team's CFP rank across the weeks,
    labeling each line by its true rank on the left side.
    """
    num_teams = len(weekly_rankings[0])

    name_to_ranks = {}
    name_to_true = {}

    for w, ranking_list in enumerate(weekly_rankings):
        for team in ranking_list:
            nm = team['name']
            if nm not in name_to_ranks:
                name_to_ranks[nm] = []
                name_to_true[nm] = team['true_rank']
            name_to_ranks[nm].append(team['cfp_rank'])

    # Generate dark colors for each team
    colors = [random_dark_color() for _ in range(num_teams)]

    fig, ax = plt.subplots(figsize=(10, 6))
    weeks_x = range(len(weekly_rankings))

    # Sort by true rank so color assignment is stable
    all_names_sorted = sorted(name_to_ranks.keys(), key=lambda n: name_to_true[n])

```

```
    for idx, nm in enumerate(all_names_sorted):
        c = colors[idx % len(colors)]
        ranks_list = name_to_ranks[nm]
        true_r = name_to_true[nm]

        ax.plot(
            weeks_x,
            ranks_list,
            marker='o',
            color=c,
            linewidth=2
        )

        # Label on left at week0
        ax.text(
            -0.2,
            ranks_list[0],
            f"T{true_r}",
            ha='right',
            va='center',
            color=c,
            fontsize=8
        )

    ax.invert_yaxis()
    ax.set_xlabel("Week")
    ax.set_ylabel("CFP Rank (1 is best)")
    ax.set_title("Team CFP Rankings (Higher Winner Rewards, Lower Loser Penalties)")
    ax.set_xticks(list(weeks_x))
    ax.set_xticklabels([f"W{w}" for w in weeks_x])
    plt.tight_layout()
    plt.show()

def main():
    weekly_data = simulate_season(num_teams=26, num_weeks=5, seed=None)

    # Print results
    for w_idx, ranking_list in enumerate(weekly_data):
        if w_idx == 0:
            print("=== Preseason (Week 0) CFP Ranking ===")
        else:
```

```
            print(f"=== Week {w_idx} CFP Ranking ===")
        for t in ranking_list:
            print(f"CFP#{t['cfp_rank']:02d} - {t['name']} "
                  f"(TrueRank={t['true_rank']}) "
                  f"- SeasonPts={t['season_points']}")
        print()

    # Plot
    plot_rankings_over_weeks(weekly_data)

if __name__ == "__main__":
    main()
```

C.3 Standard Full-FBS 100-Run Inverse Model

```python
import random
import copy
import matplotlib.pyplot as plt
import colorsys

# ============================================================
#   1) Simulation Parameters
# ============================================================
DEFAULT_NUM_TEAMS = 134
DEFAULT_NUM_WEEKS = 12
DEFAULT_RUNS = 100  # 100 runs for averaging

# ============================================================
#   2) Generate Teams
# ============================================================
def generate_teams(num_teams=DEFAULT_NUM_TEAMS):
    """
    Each team: {
        'name': e.g. "Team #1", ..., "Team #134",
        'true_rank': 1..134 (1=best),
        'cfp_rank': 1..134 (1=top in CFP),
        'season_points': 0
    }
    Inverting the preseason CFP so the best team (true_rank=1) gets cfp_rank=134, etc.
    """
    teams = []
    for i in range(1, num_teams + 1):
        true_rank = i
        # Invert for initial CFP: best => cfp_rank=134, worst => cfp_rank=1
        cfp_rank = num_teams - i + 1
        t = {
            'name': f"Team #{i}",
            'true_rank': true_rank,
            'cfp_rank': cfp_rank,
            'season_points': 0
        }
        teams.append(t)
    return teams

```

```python
# ==============================
#   3) Probability of Win
# ==============================
def probability_of_win(team_a_true, team_b_true):
    """
    FBS-like logic:
      Let diff = (team_b_true - team_a_true).
      If diff>0 => team_a is better => base_prob for team_a
      If diff<0 => team_a is worse => 1 - base_prob

      Bins (abs_diff):
       <= 5       => 50/50
        6-10      => 65/35
        11-15     => 75/25
        16-25     => 85/15
        26-50     => 95/5
        51-100    => 98/2
        >100      => 99/1
    """
    diff = team_b_true - team_a_true
    abs_diff = abs(diff)

    if abs_diff <= 5:
        base_prob = 0.50
    elif abs_diff <= 10:
        base_prob = 0.65
    elif abs_diff <= 15:
        base_prob = 0.75
    elif abs_diff <= 25:
        base_prob = 0.85
    elif abs_diff <= 50:
        base_prob = 0.95
    elif abs_diff <= 100:
        base_prob = 0.98
    else:
        base_prob = 0.99

    if diff > 0:
        return base_prob
    else:
        return 1 - base_prob

# ==============================
```

```
#   4) Determine CFP Points
# ==============================
def determine_cfp_points(team_cfp_rank, opponent_cfp_rank, did_win):
    """
    New CFP system:
      - 5 pts: Win vs. stronger team OR up to 7 spots below
      - 4 pts: Win vs. 8  24   spots below
      - 3 pts: Win vs. 25+ below OR lose to stronger team
      - 2 pts: Lose to a team 1  7   spots below
      - 1 pts: Lose to a team 8  24   spots below
      - 0 pts: Lose to a team 25+ below
    """
    if did_win:
        diff = opponent_cfp_rank - team_cfp_rank
        if opponent_cfp_rank < team_cfp_rank or diff <= 7:
            return 5
        elif diff <= 24:
            return 4
        else:
            return 3
    else:
        if opponent_cfp_rank < team_cfp_rank:
            return 3
        else:
            diff = opponent_cfp_rank - team_cfp_rank
            if diff <= 7:
                return 2
            elif diff <= 24:
                return 1
            else:
                return 0

# ==============================
#   5) Tie-Break
# ==============================
def break_ties(teams_sorted, teams_last_week):
    """
    Sort by season_points desc; if tie, keep last week's order
    """
    name_to_idx = {t['name']: i for i, t in enumerate(teams_last_week)}
    teams_sorted.sort(key=lambda t: name_to_idx[t['name']])
```

```
    teams_sorted.sort(key=lambda t: t['season_points'], reverse=True)
    return teams_sorted

# ================================
#  6) Single-Season Simulation
# ================================
def simulate_single_season(num_teams=DEFAULT_NUM_TEAMS, num_weeks=DEFAULT_NUM_WEEKS, seed=None):
    """
    Returns weekly_rankings: list of length num_weeks+1,
    each element is a deep copy of the teams in sorted CFP order for that week.
    """
    if seed is not None:
        random.seed(seed)
    else:
        random.seed()

    teams = generate_teams(num_teams)
    # Preseason CFP order
    current_cfp_order = sorted(teams, key=lambda t: t['cfp_rank'])
    weekly_rankings = [copy.deepcopy(current_cfp_order)]

    for w in range(1, num_weeks + 1):
        indices = list(range(num_teams))
        random.shuffle(indices)
        matchups = [(indices[i], indices[i+1]) for i in range(0, num_teams, 2)]

        # map from name->last week's CFP rank
        last_week_map = {t['name']: (idx+1) for idx, t in enumerate(current_cfp_order)}

        for idx_a, idx_b in matchups:
            team_a = teams[idx_a]
            team_b = teams[idx_b]
            p_a_wins = probability_of_win(team_a['true_rank'], team_b['true_rank'])
            a_wins = (random.random() < p_a_wins)

            cfp_a = last_week_map[team_a['name']]
            cfp_b = last_week_map[team_b['name']]
```

```python

            pts_a = determine_cfp_points(cfp_a, cfp_b, a_wins)
            pts_b = determine_cfp_points(cfp_b, cfp_a, not a_wins)

            team_a['season_points'] += pts_a
            team_b['season_points'] += pts_b

        # Re-sort
        teams_sorted = sorted(teams, key=lambda t: t['season_points'], reverse=True)
        new_cfp_order = break_ties(teams_sorted, current_cfp_order)

        # Update cfp_rank
        for rank_pos, tdict in enumerate(new_cfp_order):
            tdict['cfp_rank'] = rank_pos + 1

        weekly_rankings.append(copy.deepcopy(new_cfp_order))
        current_cfp_order = new_cfp_order

    return weekly_rankings

# ============================
#  7) Compute Weekly Stats
# ============================
def compute_weekly_stats(weekly_rankings):
    """
    Returns 4 lists (each length = len(weekly_rankings)):
      avg_diff[w]      = average of |cfp_rank - true_rank| at week w
      max_diff[w]      = max of |cfp_rank - true_rank| at week w
      biggest_rise[w]  = largest improvement (old_rank - new_rank) from w-1 to w
      biggest_fall[w]  = largest drop  (new_rank - old_rank) from w-1 to w
                         (# of spots dropped as a positive integer)
    For w=0, biggest_rise=0, biggest_fall=0 since there's no previous week.
    """
    num_weeks = len(weekly_rankings)
    avg_diff = [0]*num_weeks
    max_diff = [0]*num_weeks
```

```python
    biggest_rise = [0]*num_weeks
    biggest_fall = [0]*num_weeks

    # We'll create name->cfp_rank for each week
    week_to_map = []
    for w, snapshot in enumerate(weekly_rankings):
        d = {team['name']: team['cfp_rank'] for team in snapshot}
        week_to_map.append(d)

        # compute avg & max
        diffs = [abs(team['cfp_rank'] - team['true_rank']) for
   team in snapshot]
        avg_diff[w] = sum(diffs)/len(diffs)
        max_diff[w] = max(diffs)

    # biggest rise/fall
    for w in range(1, num_weeks):
        map_prev = week_to_map[w-1]
        map_this = week_to_map[w]

        best_improvement = 0
        worst_drop = 0
        for name in map_this:
            old_rank = map_prev[name]
            new_rank = map_this[name]
            movement = old_rank - new_rank
            if movement > best_improvement:
                best_improvement = movement
            drop = new_rank - old_rank
            if drop > worst_drop:
                worst_drop = drop

        biggest_rise[w] = best_improvement
        biggest_fall[w] = worst_drop

    return avg_diff, max_diff, biggest_rise, biggest_fall

# ========================================
#  8) Multiple Runs & Aggregation
# ========================================
def run_multiple_simulations(num_runs=DEFAULT_RUNS,
                             num_teams=DEFAULT_NUM_TEAMS,
                             num_weeks=DEFAULT_NUM_WEEKS):
```

```
    """
    Run the simulation `num_runs` times.
    For each run, compute the 4 weekly stats arrays.
    Then average them across all runs.

    Returns (avg_avg_diff, avg_max_diff, avg_biggest_rise, avg_biggest_fall)
    each is a list of length (num_weeks+1).
    """
    all_avg_diffs = []
    all_max_diffs = []
    all_biggest_rise = []
    all_biggest_fall = []

    for _ in range(num_runs):
        weekly_rankings = simulate_single_season(num_teams, num_weeks, seed=None)
        avg_diff, max_diff, biggest_rise, biggest_fall = compute_weekly_stats(weekly_rankings)

        all_avg_diffs.append(avg_diff)
        all_max_diffs.append(max_diff)
        all_biggest_rise.append(biggest_rise)
        all_biggest_fall.append(biggest_fall)

    weeks_count = num_weeks + 1
    avg_avg_diff = [0]*(weeks_count)
    avg_max_diff = [0]*(weeks_count)
    avg_rise = [0]*(weeks_count)
    avg_fall = [0]*(weeks_count)

    for w in range(weeks_count):
        sum_avg_d = sum(run[w] for run in all_avg_diffs)
        sum_max_d = sum(run[w] for run in all_max_diffs)
        sum_rise  = sum(run[w] for run in all_biggest_rise)
        sum_fall  = sum(run[w] for run in all_biggest_fall)

        avg_avg_diff[w] = sum_avg_d / num_runs
        avg_max_diff[w] = sum_max_d / num_runs
        avg_rise[w] = sum_rise / num_runs
        avg_fall[w] = sum_fall / num_runs

    return avg_avg_diff, avg_max_diff, avg_rise, avg_fall
```

```python

# ===========================
# 9) Plot Aggregated Stats
# ===========================
def plot_aggregated_stats(avg_avg_diff, avg_max_diff,
    avg_biggest_rise, avg_biggest_fall, num_runs):
    """
    Takes four lists (each length = num_weeks+1),
    and plots them in four separate line plots, weeks on x-axis.
    'num_runs' is used to clarify the title: "Averaged Over X Runs
    ".
    """
    weeks_count = len(avg_avg_diff)
    x_vals = list(range(weeks_count))
    x_labels = [f"W{w}" for w in x_vals]

    # 1) Average Discrepancy
    plt.figure(figsize=(8,5))
    plt.plot(x_vals, avg_avg_diff, marker='o', label='Avg
    Discrepancy')
    plt.title(f"Average |CFP Rank - True Rank| (Averaged Over {
    num_runs} Runs)")
    plt.xlabel("Week")
    plt.ylabel("Average Discrepancy")
    plt.xticks(x_vals, x_labels)
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    # 2) Maximum Discrepancy
    plt.figure(figsize=(8,5))
    plt.plot(x_vals, avg_max_diff, marker='o', color='red', label=
    'Max Discrepancy')
    plt.title(f"Maximum |CFP Rank - True Rank| (Averaged Over {
    num_runs} Runs)")
    plt.xlabel("Week")
    plt.ylabel("Maximum Discrepancy")
    plt.xticks(x_vals, x_labels)
    plt.grid(True)
    plt.legend()
    # Force y-axis to start at 0
    plt.ylim(bottom=0)
```

```
    plt.show()

    # 3) Biggest Rise
    plt.figure(figsize=(8,5))
    plt.plot(x_vals, avg_biggest_rise, marker='o', color='green', label='Biggest Rise')
    plt.title(f"Biggest Rise in Rank (Averaged Over {num_runs} Runs)")
    plt.xlabel("Week")
    plt.ylabel("Number of Spots Gained")
    plt.xticks(x_vals, x_labels)
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    # 4) Biggest Fall
    plt.figure(figsize=(8,5))
    plt.plot(x_vals, avg_biggest_fall, marker='o', color='orange', label='Biggest Fall')
    plt.title(f"Biggest Fall in Rank (Averaged Over {num_runs} Runs)")
    plt.xlabel("Week")
    plt.ylabel("Number of Spots Dropped")
    plt.xticks(x_vals, x_labels)
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

# ============================
# 10) Main
# ============================
def main():
    num_runs = 100
    num_teams = 134
    num_weeks = 12

    print(f"Running {num_runs} simulations of {num_teams} teams for {num_weeks} weeks each...")

    (avg_avg_diff, avg_max_diff,
     avg_biggest_rise, avg_biggest_fall) =
```

```
    run_multiple_simulations(
            num_runs=num_runs,
            num_teams=num_teams,
            num_weeks=num_weeks
    )

    # Print out the weekly data points
    print(f"\n=== Weekly Averages Over {num_runs} Runs ===")
    print(f"{'Week':<4} | {'AvgDiff':>8} | {'MaxDiff':>8} | {'
    MaxRise':>8} | {'MaxFall':>8}")
    print("-"*46)
    weeks_count = num_weeks + 1

    for w in range(weeks_count):
        print(f"{w:<4d} | {avg_avg_diff[w]:8.2f} | "
              f"{avg_max_diff[w]:8.2f} | "
              f"{avg_biggest_rise[w]:8.2f} | "
              f"{avg_biggest_fall[w]:8.2f}")

    # Now plot the aggregated results
    plot_aggregated_stats(avg_avg_diff, avg_max_diff,
    avg_biggest_rise, avg_biggest_fall, num_runs)

if __name__ == "__main__":
    main()
```

C.4 Harsher Full-FBS 100-Run Inverse Model

```python
import random
import copy
import matplotlib.pyplot as plt


# ==================================================
#   1) Simulation Parameters
# ==================================================
DEFAULT_NUM_TEAMS = 134
DEFAULT_NUM_WEEKS = 12
DEFAULT_RUNS = 100  # 100 runs to remove statisitcal anomalies

# ==================================================
#   2) Generate Teams
# ==================================================
def generate_teams(num_teams=DEFAULT_NUM_TEAMS):
    """
    Each team: {
      'name': e.g. "Team #1", ..., "Team #134",
      'true_rank': 1..134 (1=best),
      'cfp_rank': 1..134 (1=top in CFP),
      'season_points': 0
    }
    Inverting the preseason CFP so the best team (true_rank=1) gets cfp_rank=134, etc.
    """
    teams = []
    for i in range(1, num_teams + 1):
        true_rank = i
        # Inverse assignment for initial CFP
        cfp_rank = num_teams - i + 1
        t = {
            'name': f"Team #{i}",
            'true_rank': true_rank,
            'cfp_rank': cfp_rank,
            'season_points': 0
        }
        teams.append(t)
    return teams

# ==================================================
```

```python
#  3) Probability of Win (Game logic)
# ==============================
def probability_of_win(team_a_true, team_b_true):
    """
    FBS-like logic:
      Let diff = (team_b_true - team_a_true).
      If diff>0 => team_a is better => base_prob for A
      If diff<0 => team_a is worse => 1 - base_prob

      Bins (abs_diff):
       <= 5      => 50/50
        6-10     => 65/35
        11-15    => 75/25
        16-25    => 85/15
        26-50    => 95/5
        51-100   => 98/2
        >100     => 99/1
    """
    diff = team_b_true - team_a_true
    abs_diff = abs(diff)

    if abs_diff <= 5:
        base_prob = 0.50
    elif abs_diff <= 10:
        base_prob = 0.65
    elif abs_diff <= 15:
        base_prob = 0.75
    elif abs_diff <= 25:
        base_prob = 0.85
    elif abs_diff <= 50:
        base_prob = 0.95
    elif abs_diff <= 100:
        base_prob = 0.98
    else:
        base_prob = 0.99

    if diff > 0:
        return base_prob
    else:
        return 1 - base_prob

# ==============================
#  4) Determine CFP Points (Harsher Variation)
```

```python
# ==============================
def determine_cfp_points(team_cfp_rank, opponent_cfp_rank, did_win):
    """
    New 'harsh' system for CFP points:

    Winners:
      - 9 points: beating a stronger team (opponent_cfp_rank < yours)
      - 8 points: beating a team up to 7 spots below (diff  7)
      - 7 points: beating a team 8 24  spots below
      - 6 points: beating a team 25+ spots below

    Losers:
      - 4 points: losing to a stronger team (opponent_cfp_rank < yours)
      - 2 points: losing to a team 1 5  spots below (1  diff  5)
      - 1 point : losing to a team 6 24  spots below (6  diff  24)
      - 0 points: losing to a team 25+ spots below (diff  25)

    Here, "lower" means opponent_cfp_rank > team_cfp_rank (a worse rank number).
    """
    diff = opponent_cfp_rank - team_cfp_rank  # if negative => opponent is stronger
    if did_win:
        # Win:
        if opponent_cfp_rank < team_cfp_rank:
            # beating a stronger team
            return 9
        elif diff <= 7:
            return 8
        elif diff <= 24:
            return 7
        else:
            return 6
    else:
        # Loss:
        if opponent_cfp_rank < team_cfp_rank:
            # lost to a stronger team
            return 4
```

```python
        else:
            if diff <= 5:
                return 2
            elif diff <= 24:
                return 1
            else:
                return 0

# ==============================
#   5) Tie-Break
# ==============================
def break_ties(teams_sorted, teams_last_week):
    """
    Sort by season_points desc; if tie, keep last week's order
    """
    name_to_idx = {t['name']: i for i, t in enumerate(teams_last_week)}
    teams_sorted.sort(key=lambda t: name_to_idx[t['name']])
    teams_sorted.sort(key=lambda t: t['season_points'], reverse=True)
    return teams_sorted

# ==============================
#   6) Single-Season Simulation
# ==============================
def simulate_single_season(num_teams=DEFAULT_NUM_TEAMS, num_weeks=DEFAULT_NUM_WEEKS, seed=None):
    """
    Returns weekly_rankings: list of length num_weeks+1,
    each element is a deep copy of the teams in sorted CFP order for that week.
    """
    if seed is not None:
        random.seed(seed)
    else:
        random.seed()

    teams = generate_teams(num_teams)
    # Preseason
    current_cfp_order = sorted(teams, key=lambda t: t['cfp_rank'])
    weekly_rankings = [copy.deepcopy(current_cfp_order)]

    for w in range(1, num_weeks + 1):
```

```
        indices = list(range(num_teams))
        random.shuffle(indices)
        matchups = [(indices[i], indices[i+1]) for i in range(0, num_teams, 2)]

        # map from name->last week's CFP rank
        last_week_map = {t['name']: (idx+1) for idx, t in enumerate(current_cfp_order)}

        for idx_a, idx_b in matchups:
            team_a = teams[idx_a]
            team_b = teams[idx_b]

            p_a_wins = probability_of_win(team_a['true_rank'], team_b['true_rank'])
            a_wins = (random.random() < p_a_wins)

            cfp_a = last_week_map[team_a['name']]
            cfp_b = last_week_map[team_b['name']]

            pts_a = determine_cfp_points(cfp_a, cfp_b, a_wins)
            pts_b = determine_cfp_points(cfp_b, cfp_a, not a_wins)

            team_a['season_points'] += pts_a
            team_b['season_points'] += pts_b

        # Re-sort
        teams_sorted = sorted(teams, key=lambda t: t['season_points'], reverse=True)
        new_cfp_order = break_ties(teams_sorted, current_cfp_order)

        # Update cfp_rank
        for rank_pos, tdict in enumerate(new_cfp_order):
            tdict['cfp_rank'] = rank_pos + 1

        weekly_rankings.append(copy.deepcopy(new_cfp_order))
        current_cfp_order = new_cfp_order

    return weekly_rankings

# ==========================
#   7) Compute Weekly Stats
```

```python
# ====================================
def compute_weekly_stats(weekly_rankings):
    """
    Returns 4 lists (each length = len(weekly_rankings)):
      avg_diff[w]      = average of |cfp_rank - true_rank| at week w
      max_diff[w]      = max of |cfp_rank - true_rank| at week w
      biggest_rise[w]  = largest improvement (old_rank - new_rank) from w-1 to w
      biggest_fall[w]  = largest drop  (new_rank - old_rank) from w-1 to w
    For w=0, biggest_rise=0, biggest_fall=0 (no prior week).
    """
    num_weeks = len(weekly_rankings)
    avg_diff = [0]*num_weeks
    max_diff = [0]*num_weeks
    biggest_rise = [0]*num_weeks
    biggest_fall = [0]*num_weeks

    # We'll create name->cfp_rank for each week for easy referencing
    week_to_map = []
    for w, snapshot in enumerate(weekly_rankings):
        d = {team['name']: team['cfp_rank'] for team in snapshot}
        week_to_map.append(d)

        # compute avg & max
        diffs = [abs(team['cfp_rank'] - team['true_rank']) for team in snapshot]
        avg_diff[w] = sum(diffs)/len(diffs)
        max_diff[w] = max(diffs)

    # biggest rise/fall
    for w in range(1, num_weeks):
        map_prev = week_to_map[w-1]
        map_this = week_to_map[w]

        best_improvement = 0  # old_rank - new_rank
        worst_drop = 0        # new_rank - old_rank
        for name in map_this:
            old_rank = map_prev[name]
            new_rank = map_this[name]
            movement = old_rank - new_rank
```

```
            if movement > best_improvement:
                best_improvement = movement
            drop = new_rank - old_rank
            if drop > worst_drop:
                worst_drop = drop

        biggest_rise[w] = best_improvement
        biggest_fall[w] = worst_drop

    return avg_diff, max_diff, biggest_rise, biggest_fall

# ==============================
#   8) Multiple Runs & Aggregation
# ==============================
def run_multiple_simulations(num_runs=DEFAULT_RUNS,
                             num_teams=DEFAULT_NUM_TEAMS,
                             num_weeks=DEFAULT_NUM_WEEKS):
    """
    Run the simulation `num_runs` times.
    For each run, compute the 4 weekly stats arrays.
    Then average them across all runs.

    Returns (avg_avg_diff, avg_max_diff, avg_biggest_rise, avg_biggest_fall)
    each is a list of length (num_weeks+1).
    """
    all_avg_diffs = []
    all_max_diffs = []
    all_biggest_rise = []
    all_biggest_fall = []

    for _ in range(num_runs):
        weekly_rankings = simulate_single_season(num_teams, num_weeks, seed=None)
        avg_diff, max_diff, biggest_rise, biggest_fall = compute_weekly_stats(weekly_rankings)

        all_avg_diffs.append(avg_diff)
        all_max_diffs.append(max_diff)
        all_biggest_rise.append(biggest_rise)
        all_biggest_fall.append(biggest_fall)

    # Now average each metric across runs
```

```
    weeks_count = num_weeks + 1
    avg_avg_diff = [0]*(weeks_count)
    avg_max_diff = [0]*(weeks_count)
    avg_rise = [0]*(weeks_count)
    avg_fall = [0]*(weeks_count)

    for w in range(weeks_count):
        sum_avg_d = sum(run[w] for run in all_avg_diffs)
        sum_max_d = sum(run[w] for run in all_max_diffs)
        sum_rise  = sum(run[w] for run in all_biggest_rise)
        sum_fall  = sum(run[w] for run in all_biggest_fall)

        avg_avg_diff[w] = sum_avg_d / num_runs
        avg_max_diff[w] = sum_max_d / num_runs
        avg_rise[w] = sum_rise / num_runs
        avg_fall[w] = sum_fall / num_runs

    return avg_avg_diff, avg_max_diff, avg_rise, avg_fall

# ================================
# 9) Plot Aggregated Stats
# ================================
def plot_aggregated_stats(avg_avg_diff, avg_max_diff, avg_biggest_rise, avg_biggest_fall, num_runs):
    """
    Takes four lists (each length = num_weeks+1),
    and plots them in four separate line plots, weeks on x-axis.
    """
    weeks_count = len(avg_avg_diff)
    x_vals = list(range(weeks_count))
    x_labels = [f"W{w}" for w in x_vals]

    # 1) Average Discrepancy
    plt.figure(figsize=(8,5))
    plt.plot(x_vals, avg_avg_diff, marker='o', label='Avg Discrepancy')
    plt.title(f"[Harsh Committee] Average |CFP - True| (Over {num_runs} Runs)")
    plt.xlabel("Week")
    plt.ylabel("Average Discrepancy")
    plt.xticks(x_vals, x_labels)
    plt.grid(True)
    plt.legend()
```

```python
    plt.tight_layout()
    plt.show()

    # 2) Maximum Discrepancy
    plt.figure(figsize=(8,5))
    plt.plot(x_vals, avg_max_diff, marker='o', color='red', label=
    'Max Discrepancy')
    plt.title(f"[Harsh Committee] Maximum |CFP - True| (Over {
    num_runs} Runs)")
    plt.xlabel("Week")
    plt.ylabel("Maximum Discrepancy")
    plt.xticks(x_vals, x_labels)
    plt.grid(True)
    plt.legend()
    plt.ylim(bottom=0)  # start y-axis at 0
    plt.show()

    # 3) Biggest Rise
    plt.figure(figsize=(8,5))
    plt.plot(x_vals, avg_biggest_rise, marker='o', color='green',
    label='Biggest Rise')
    plt.title(f"[Harsh Committee] Biggest Rise in Rank (Over {
    num_runs} Runs)")
    plt.xlabel("Week")
    plt.ylabel("Number of Spots Gained")
    plt.xticks(x_vals, x_labels)
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
    plt.show()

    # 4) Biggest Fall
    plt.figure(figsize=(8,5))
    plt.plot(x_vals, avg_biggest_fall, marker='o', color='orange',
     label='Biggest Fall')
    plt.title(f"[Harsh Committee] Biggest Fall in Rank (Over {
    num_runs} Runs)")
    plt.xlabel("Week")
    plt.ylabel("Number of Spots Dropped")
    plt.xticks(x_vals, x_labels)
    plt.grid(True)
    plt.legend()
    plt.tight_layout()
```

```python
    plt.show()

# ==============================
# 10) Main
# ==============================
def main():
    num_runs = 100
    num_teams = 134
    num_weeks = 12

    print(f"Running {num_runs} simulations [Harsh Committee Variation] with {num_teams} teams for {num_weeks} weeks each...")

    (avg_avg_diff, avg_max_diff,
     avg_biggest_rise, avg_biggest_fall) = run_multiple_simulations(
            num_runs=num_runs,
            num_teams=num_teams,
            num_weeks=num_weeks
    )

    # Print out the weekly data points
    print("\n=== Weekly Averages (Harsh Committee) Over 100 Runs ===")
    print(f"{'Week':<4} | {'AvgDiff':>8} | {'MaxDiff':>8} | {'MaxRise':>8} | {'MaxFall':>8}")
    print("-"*46)
    weeks_count = num_weeks + 1

    for w in range(weeks_count):
        print(f"{w:<4d} | {avg_avg_diff[w]:8.2f} | "
              f"{avg_max_diff[w]:8.2f} | "
              f"{avg_biggest_rise[w]:8.2f} | "
              f"{avg_biggest_fall[w]:8.2f}")

    # Now plot the aggregated results
    plot_aggregated_stats(avg_avg_diff, avg_max_diff, avg_biggest_rise, avg_biggest_fall, num_runs)

if __name__ == "__main__":
    main()
```