Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes] The abstract and introduction provide representative summaries of our work.
   (b) Did you describe the limitations of your work? [Yes] See section 4.1.2 and the Discussion.
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results...
   (a) Did you state the full set of assumptions of all theoretical results? [N/A]
   (b) Did you include complete proofs of all theoretical results? [N/A]
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [Yes]
   (b) Did you specify all the training details (e.g., data splits, hyperparameters, how they were chosen)? [Yes] See methods and Appendix D.
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes] See section F.
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [N/A]
   (b) Did you mention the license of the assets? [Yes] See Appendix G.
   (c) Did you include any new assets either in the supplemental material or as a URL? [Yes]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [N/A]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [N/A]
5. If you used crowdsourcing or conducted research with human subjects...
   (a) Did you include the full text of instructions given to participants and screenshots, if applicable? [N/A]
   (b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable? [N/A]
   (c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation? [N/A]

Appendix

A Notation

The following notation is used throughout this paper:

- n the number of players in the game
- G the number of games
- M the maximum sequence length in variable feedforward networks
- q the quota, threshold that determines when a coalition can complete the goal or task
- w_i the weight of a player i
- $x_i = w_i/q$, the normalized weight of a player i
- N the grand coalition
- C a coalition (subset of players)
- C^{min} the minimal set of winning coalitions: the coalitions where each player is pivotal
- $\mathbf{X}$ feature matrix
- $\mathbf{P}$ solution matrix
- p_i the allocated reward or payoff to a player i
- ε the least core value (LCV), i.e. joint payoff to sacrifice such that there exists a feasible payoff vector in the least core

B Motivating The Use Of The Shapley Value To Analyze Influence In Weighted Voting Games

We provide an intuitive example motivating the use of Shapley values for measuring the true influence of participants, in weighted voting games or political power in decision making bodies.

Consider a parliament with 100 seats, and imagine that following the elections we have two large parties of 49 seats each, and one tiny party who has only managed to get 2 seats. However, note that having a majority means controlling a majority of the seats, i.e. more than 50 seats. This means that neither of the two big parties has a majority on its own (and of course the small party hasn't got a majority on its own).

Hence, to achieve a required majority and form a government or pass a bill, the parties have to work in teams. Clearly, a coalition consisting of the two big parties has a majority, as together they have 49+49=98 seats out of the total of 100. However, a coalition consisting of any of the two big parties and the one small party also has 49+2=51 seats, which is also a majority.

Hence, in the setting discussed above, *any* two parties form a winning coalition that has a majority and is able to form a government or pass a bill. This can be modelled as a weighted voting game, where the weights are $(w_1 = 49, w_2 = 49, w_3 = 2)$ and the quota is $q = 50$.

Intuitively, as any two parties or more are a winning team and as any single party is a losing team, we might say that the weights are irrelevant. Although the big parties each have almost 25 times the seats as the small party, they all are symmetric in their opportunities to form a winning coalition, and one could claim they actually have equal political power. Clearly, the small party is in a very strong negotiation position, and could demand for instance to control a significant proportion of the joint budget.

In the above weighted voting game, the Shapley value of all agents would be equal (each getting a value of $\frac{1}{3}$), as it considers the marginal contributions of the parties over all permutations of players. Due to the symmetry between the parties, they have identical marginal contributions.

The above specific example illustrates some of the principles behind the fairness axioms that the Shapley value exhibits, and illustrates its importance for measuring influence in decision making bodies.

C Algorithm For Generating least core Solutions

The least core solutions can be generated by considering all possible coalitions. A simple improvement, however, is to use the minimal set of winning coalitions [16] instead. In this section, we describe the idea behind this approach and the algorithm used in this paper.

Let $N \equiv \{1, 2, \ldots, n\}$ denote the set of all $N \in \mathbb{Z}_{\geq 0}$ players, let $q \in \mathbb{R}_{\geq 0}$ denote the quota and let $w_i \in \mathbb{R}_{\geq 0}$ denote the weights for player i, for all $i \in N$. The characteristic function is given by $v(C) \in [0, 1]$, for coalitions $C \subseteq N$.

The naive approach to find the least core solutions is by considering all winning coalitions C^{win} defined as

$$C^{\text{win}} \equiv \left\{ C \subseteq N \;\middle|\; C \neq \varnothing \text{ and } \sum_{i \in C} w_i \geq q \right\} \tag{C.1}$$

and to find a least core solution by solving LP-problem C.1a. In the LP-problem (C.1a), all winning

$$\begin{aligned} \min_{p_1, p_2, \ldots, p_n, \varepsilon} \quad & \varepsilon \\ \text{s.t.} \quad & \sum_{i \in C} p_i \geq 1 - \varepsilon \quad \forall C \in C^{\text{win}} \\ & \sum_{i \in N} p_i = 1 \\ & p_i \geq 0 \qquad \forall i \in N \end{aligned}$$

(a) Naive approach

$$\begin{aligned} \min_{p_1, p_2, \ldots, p_n, \varepsilon} \quad & \varepsilon \\ \text{s.t.} \quad & \sum_{i \in C} p_i \geq 1 - \varepsilon \quad \forall C \in C^{\min} \\ & \sum_{i \in N} p_i = 1 \\ & p_i \geq 0 \qquad \forall i \in N \end{aligned}$$

(b) Minimal approach

Figure C.1: Algorithms for finding least core solutions.

coalitions are considered, Inside those winning coalitions there exists a subset, that could branch out and form a smaller coalition. This yields a higher payoff for the players in the sub-coalition and is hence a Pareto improvement for the "decision-makers". Therefore, we can instead only consider the subset of *minimal winning* coalitions, given by

$$C^{\min} \equiv \left\{ C \subseteq N \;\middle|\; C \neq \varnothing, \sum_{i \in S} w_i \geq q, \text{ and, for all } j \in C, \sum_{i \in C \setminus \{j\}} w_i < q \right\} \tag{C.2}$$

and solve the more efficient LP-problem C.1b.

D Experimental details

In this section, we elaborate on the experimental procedures, model architectures and hyperparameters used in our experiments.

Code and notebook. Following the guidelines, we provide python code to reproduce our results, a pedagogical notebook (called CooperativeGameTheoryPrimer.ipynb) that introduces the most important game theory formalisms and the necessary data in the accompanied zip file.

D.1 Weighted Voting Games

Optimization procedure fixed-size feedforward networks. For each N-player dataset $\mathcal{D}^N_{\text{fixed}}$ we perform $R = 100$ independent runs with a maximum of $E = 6000$ epochs and an early-stopping criteria which breaks a run if, after a baseline of 500 epochs, the validation loss does not improve for 75 consecutive epochs. For each run, 70 percent is allocated for training and 30 percent for validation. After training, we select the model with the best validation loss. The selected models are stored and used for evaluation. In all experiments we use the Adam [29] optimizer, ReLU [43] activation functions for the hidden layers, Softmax functions for output layer of the payoffs and a Sigmoid for the output layer of epsilon.

Optimization procedure var-size feedforward network. We perform $R = 100$ independent runs for one variable dataset $\mathcal{D}_{\text{var}}$ with a maximum of $E = 15000$ number of epochs and an early-stopping criteria which breaks a run if, after a baseline of 500 epochs, the validation loss does not improve for 75 consecutive epochs. For each run, 70 percent is allocated for training and 30 percent for validation. After training, we select the model with the best validation loss, which is stored and used

for evaluation. In all experiments we use the Adam [29] optimizer, ReLU [43] activation functions for the hidden layers, Softmax functions for output layer of the payoffs and a Sigmoid for the output layer of epsilon.

Hyperparameters. All results are generated with multi-layer perceptrons (feedforward networks) of varying hidden-layer size and width. Further hyperparameters are the dropout rate, weight decay in the Adam optimizer, and the learning rate. Our standard model architectures are depicted in Figure 3 Right.

Approximated ground-truth Shapley values. To obtain the ground-truth solutions for Shapley values above nine players, we perform Monte-Carlo sampling to obtain a subset of all possible permutations. For each of the $R = 10$ resamples we randomly sample $P = 1000$ permutations and average across the resamples to obtain approximations of the ground-truth. To validate that the approximated labels are representative we compute the Mean MAE between the true and approximated Shapley values for $n = 5$ up to $n = 10$ players with 1000 games per n-player game. Across all n-player games the mean MAE between the true and approximated Shapley values was 0.0011 and never exceeded 0.0014. Figure D.2 shows the distributions per player together with the true label (red dot) for 8 and 9 player games.

Parameterizations for test datasets. Figure D.1 shows the five standardized test distributions that are generated with the parameters in Table 2.

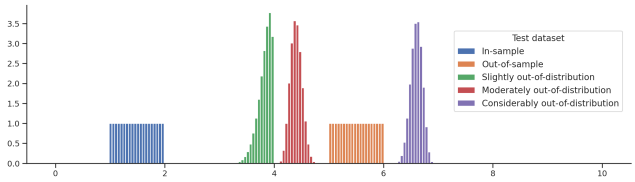

Figure D.1: **Test data distributions in perspective.** To assess how robust models' are to shifts in the weight distribution, we re-parameterize the weight distribution in five different ways.

Table 2: Test distributions are generated by the following parameterizations of the beta distribution

Test set name	α	β	location
In-sample	1	1	1
Out-of-sample	1	1	2.5N
Slightly out-of-distribution	8	12	2
Moderately out-of-distribution	7	1.5	1.5N
Considerably out-of-distribution	12	8	3N

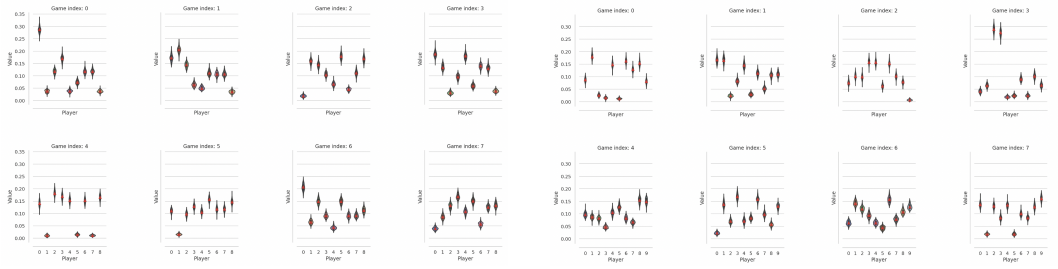

Figure D.2: **Approximating ground-truth Shapley values.** Ten randomly sampled games with distribution for each player in the game, the red dot indicates the true Shapley value (i.e. when computing the marginal value for each permutation).

D.2 Explainable AI

We provide a step-wise explanation of how we performed the experiment in our explainable AI section, that is, train a neural network to predict the Shapley values for a dataset of features:

1. **Select a dataset.** The first step is to select a dataset of choice. In our example, we end up with a feature matrix $\mathbf{X}^{N\times F}$ where each row is an observation and each column a feature, and a target vector $\mathbf{y} \in \mathbb{R}^N$ or $y \in [0, 1]$ depending on the task.
2. **Feature to target.** Following, we estimate a model $f_{\boldsymbol{\theta}} : \mathbb{R}^F \to \mathbb{R}$ that predicts one output for each set of features.
3. **Compute Shapley values through sampling.** We use the trained model from step 2 to obtain Shapley values with the SHAP package KernelExplainer[35], which is based on LIME [35]. This yiels a matrix of Shapley values, one for each feature in every data instance: $\boldsymbol{\Phi} \in \mathbb{R}^{N\times F}$.
4. **Feature to Shapley.** Finally, we create an array of $T = 999$ linearly spaced train-test splits $\mathbf{t} = (0.001, 0.002, \ldots, 0.998, 0.999)$. Each element t_i in $\mathbf{t}$ indicates the ratio of samples that are used for training the neural network, with the remaining $(1-t_i)N$ samples used for evaluation. For each train-test partition in $\mathbf{t}$ then, we train a feed forward neural network on t_iN samples by minimizing the MSE between the predicted and actual Shapley values (full details in Table 3) for 100 epochs, and evaluate on the $1 - t_iN$ remainder (unseen) samples.

Datasets and preprocessing. Table 4 summarizes the considered datasets. The UCI Banking dataset contains 15 features and a binary target (whether the client has subscribed a term deposit), whereas the Melbourne housing dataset contains 21 features and a regression target (the price of a house). We exclude the "duration" feature in the Banking dataset to prevent data leakage, as is recommended in the documentation. Moreover, we exclude the "Rooms" feature as there is a lot of overlap in this and the "Bedroom2" feature. The numerical values in both datasets are z-scored, and categorical features are encoded as numbers. Missing values are encoded as zeros, so that the complete dataset can be used for our experiment.

Feature to target. We partition the dataset into train and test and train a Random Forest Classification model on the UCI Banking set, obtaining a final test accuracy of 72 %. Moreover, we train a decision tree regression model to predict the price given the set of house features and obtain a final MSE of 1.21 on the test dataset.

Compute Shapley values. We use the trained models to obtain Shapley values for each instance in the feature dataset $\mathbf{X}$ using the SHAP kernel explainer object [35].

Feature to Shapley: procedure. Finally, we can train a neural network using the datasets $\mathbf{X}, \boldsymbol{\Phi}$. To investigate the sample complexity required to learn a representative mapping from features to Shapley values, we conduct the following experiment. First, we create an array of $T = 999$ linearly spaced training splits $\mathbf{t} = (0.001, 0.002, \ldots, 0.998, 0.999)$. Each element in t_i in $\mathbf{t}$ indicates the ratio of samples that are used for training the neural network, leaving the remaining $(1 - t_i)N$ samples for evaluation. For each train/test partition in $\mathbf{t}$ then, train a feed forward neural network on t_iN samples through minimizing the MSE between the predicted and actual Shapley values (architecture and hyperparameters specified in Table 3) for 100 epochs, and evaluate on the $1 - t_iN$ samples. Figure 9 depicts the resulting RMSE for both datasets as a function of the training fraction.

Table 3: Hyperparameters used for XAI experiments.

Hyperparameter	
Number of layers	3
Hidden size	128
Adam learning rate	1e-4
Adam ϵ	1e-5
Dropout rate	0.1

Table 4: Number of features pre and post processing for each considered dataset.

Dataset	Number of features		Number of observations	
	Pre	Post	Pre	Post
UCI Banking [42]	15	14	11162	11162
Melbourne Housing [47]	20	20	34857	34857

E Additional Results

In-sample predictive performance. For completeness, we provide the MAEs for each solution concepts on the in sample dataset up to $n = 15$ players in Table 5.

Table 5: Predictive performance across solution concepts on in-sample test dataset.

	least core payoffs		**least core excess**		**Shapley values**		**Banzhaf indices**	
Metric	Mean MAE		MAE		Mean MAE		Mean MAE	
N	Fixed	Variable	Fixed	Variable	Fixed	Variable	Fixed	Variable
4	0.087	0.140	0.034	0.058	0.069	0.072	0.067	0.097
5	0.077	0.096	0.031	0.045	0.040	0.047	0.035	0.060
6	0.058	0.069	0.028	0.033	0.026	0.031	0.021	0.037
7	0.040	0.051	0.021	0.027	0.022	0.023	0.014	0.024
8	0.027	0.036	0.015	0.021	0.016	0.018	0.015	0.017
9	0.019	0.031	0.012	0.019	0.010	0.015	0.012	0.014
10	0.014	0.023	0.007	0.017	0.009	0.013	0.006	0.013
11	0.010	0.021	0.006	0.020	0.006	0.012	0.009	0.013
12	0.009	0.018	0.005	0.023	0.005	0.011	0.004	0.013
13	0.006	0.016	0.003	0.027	0.009	0.010	0.004	0.014
14	0.005	0.014	0.003	0.032	0.004	0.009	0.009	0.014
15	0.005	0.013	0.003	0.036	0.004	0.009	0.012	0.014
Avg	0.019	0.015	0.029	0.051	0.018	0.022	0.017	0.027

Capturing step-wise jumps: full description. We provide a more detailed description to Figure 7. In the Figure on the **left**, each player is a winning coalition by itself when $q = 1$. Up to $q = 3$, player 3 is more critical than players 1 and 2 and receives a larger share of the joint payoffs. All players obtain equal payoffs when the only winning coalition is the grand coalition. On the **right** side: each player is a winning coalition by itself when $q = 1$, which results in an equal payoff of 0.2 for each player. Increasing the quota continuously alters the set of winning coalitions, thereby the relative importance of each player. At $q = 3$ players 1, 2 and 3 have become less critical and obtain a payoff of 0.18 each compared to 0.22 for player 4 and 5. Finally, when $q = \sum_{i=1}^{N} w_i = 7$, the only winning coalition that can be formed is the grand coalition: each player is required and the joint payoff is divided equally.

F Hardware details

We trained our neural networks on an internal compute cluster with 10 Quadro RTX 6000 GPUs and two Intel(R) Xeon(R) Gold 5218 CPUs, each of which has 16 physical cores (32 logical processors).

G Asset licensing

Two main assets were used in this work:

- The UCI Banking dataset
- The Melbourne Housing dataset

The Melbourne Housing dataset was released under the license CC BY-NC-SA 4.0. The UCI Banking dataset is licensed under a Creative Commons Attribution 4.0 International (CC BY 4.0) license.

H Heuristics and Simpler Models

We now consider a common heuristic for payoff allocation in weighted voting games, the *weight proportional* payoff allocation. We use this as a benchmark to evaluate our models' performance against. We find that our fixed models outperform this benchmark heuristic across solution concepts and test sets.

WeightProportional baseline. The WeightProportional (WP) heuristic divides the payoffs in a way that is directly proportional to the players' weights, that is, the payoff of a player j is:

$$\mathbf{p}_j = \frac{w_j}{\sum_{i=1}^{n} w_i} \tag{H.1}$$

for any $n-$player game. WP provides meaningful benchmark as it is the most naive method for allocating payoffs. The resulting payoffs (and thus errors) do not take into account the quota, which may have a major impact on the correct solutions by means of the emergent set of winning coalitions. As

such, any improvement on this benchmark thus suggests a better understanding of the collaborative structures in the game at hand.

We also compare the predictive performance of our models against the predictive performance of multinomial regression models. These models have just one layer with n inputs and n outputs for each $n-$player game and a softmax nonlinearity at the end. Across all game sizes, our models outperform the linear models by 45 % to 95 % percent. Results are found in Table 6.

Table 6: Comparing neural network performance to multinomial logistic regression models.

	Linear Core	Neural net Core	**Linear Shapley**	Neural net Shapley	**Linear Banzhaf**	Neural net Banzhaf
N						
4	0.212	0.089	0.137	0.07	0.116	0.069
5	0.147	0.078	0.098	0.041	0.083	0.035
6	0.124	0.059	0.080	0.026	0.071	0.021
7	0.108	0.041	0.070	0.022	0.063	0.014
8	0.102	0.028	0.065	0.015	0.059	0.014
9	0.089	0.020	0.056	0.009	0.051	0.012
10	0.078	0.014	0.050	0.008	0.045	0.006
11	0.072	0.010	0.045	0.006	0.042	0.009
12	0.065	0.009	0.041	0.005	0.041	0.004
13	0.062	0.006	0.036	0.009	0.035	0.004
14	0.056	0.005	0.036	0.004	0.034	0.009
15	0.057	0.005	0.032	0.004	0.032	0.0012
16	0.051	0.003	0.029	0.004	0.030	0.003
17	0.051	0.003	0.028	0.004	0.029	0.006
18	0.047	0.005	0.028	0.007	0.027	0.011
19	0.045	0.002	0.026	0.007	0.025	0.003
20	0.044	0.003	0.025	0.006	0.024	0.006

I Hard cases

Across solution concepts (Shapley, Banzhaf and least core), discontinuous jumps in the solution space yield large errors (see Section 4.1.2 and Figures 7, J.4). In this section, we highlight failure cases for the least core. In contrast to the power indices, a solution in the least core is obtained by solving a Linear Program under several hard constraints. Thus, a solution to a weighted voting game is only feasible when, for each winning coalition $C \in C^{\text{win}}$, we have that

$$v(C) = \sum_{i=1}^{C} p_i \geq 1 - \varepsilon \tag{I.1}$$

where p_i is the payoff of player i in coalition C. In words, each winning coalition must obtain joint reward that is at least as good as what they would have obtained on their own. If this criteria is not met, the solution is infeasible: some players will have an incentive to deviate from the grand coalition.

Majority of the model predictions fail to meet hard constraints. Figure I.1 depicts the percentage of feasible solutions over the game sizes. For small player games ($n < 8$), between 50 and 5.5 % of the model predictions are feasible). The larger the number of players, the fewer model predictions satisfy the hard constraints. From eight player games and above, the number of feasible predictions ranges between 0.2 and 10 %. A preliminary analysis shows no relationship between infeasible solutions and larger errors in the payoffs nor the LCV.

The fact that the model is not able to predict many feasible solutions in these large games is unsurprising. The set of winning coalitions increases approximately exponentially in n. As such, the more players in a game, the more hard constraints need to be satisfied in order for a solution to be feasible per game. All hard constraints need to be met in order for a solution to be feasible. We expect that including a penalty term for infeasible predictions during training will help to improve this metric and hope to address this in future work.

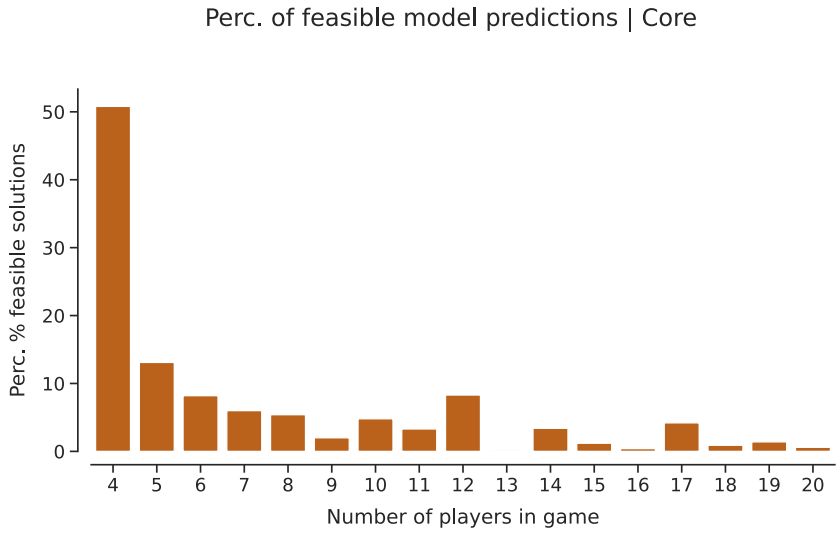

Figure I.1: **Percentage of feasible solutions for each $n-$player game**. We count the number of solutions that meet the hard constraints by validating equation I.1 for each winning coalition C in the set of winning coalitions across 1000 samples per $n-$player game.

When the least core is a set, simplex based methods tend to find corner solutions, which leads to ambiguity in the solutions. Further analysis of individual weighted voting games reveals that the most problematic games are cases where the least core contains multiple correct solutions, but the full joint payoff is allocated to a single player. In other words, payoffs are divided such that one player receives one, and all the others zero. To illustrate what is happening here, consider a simple four player game that produced the largest error (Figure I.2). We focus our analysis on the game on the left, but our insights apply to a wide range of n-player games in the dataset. This particular game has the following weights and quota (rounded to one decimal place for simplicity)

$$\mathbf{w} = (2.8, 1.6, 6.6, 1.5); \quad q = 12.1 \tag{I.2}$$

where we can see that no player is a winning coalition by itself (all w_i's $<$ q). In fact, the only winning coalition in this game is the grand coalition, that is,

$$C^{\text{min}} = \{1, 2, 3, 4\} \tag{I.3}$$

The solution to this weighted voting game is:

$$\mathbf{p} = (1, 0, 0, 0) \quad \text{with} \quad \text{LCV} = 0 \tag{I.4}$$

where player one receives the full joint payoff, even while its' weight is smaller than the weight of player three ($2.8 < 6.6$). The reason why we obtain this solution comes down to two facts. First of all, the least core is a set, and can therefore contain multiple correct solutions. Secondly, simplex based methods return corner solutions. Thus, in cases where several players are required to form a winning coalition, the solver will allocate the joint payoff to an arbitrary player that belongs to this critical subset. Figure I.3 depicts this visually for a simple two player game. Here, we have the WVG $\mathbf{w} = (1, 1)$ with $q = 2$, so the only winning coalition is the grand coalition. Solving this game results in either of the corner solutions: the payoff vector $\mathbf{p} = (1, 0)$ or $\mathbf{p} = (0, 1)$ with a LCV $= 0$. However, if we define α as the solution of a player one, any combination $\mathbf{p} = (\alpha, 1 - \alpha)$ is a feasible solution in the least core.

Obtaining the full set of solutions for this game can be done empirically. In principle, we can take the payoff vector and LCV above, generate the set of permutations of length two (player pairs) and check whether the solution is still feasible if we give a small value δ from one player (whose payoff is not zero) to any other player. However, this approach does not scale to larger games. It would be more efficient to derive rules for acquiring the full least core set from a single solution.

We limited our analysis above to four-player games. However, we observe the above behavior irrespective of the number of players in the game. Therefore, addressing this issue is critical for improving the predictive performance of our models.

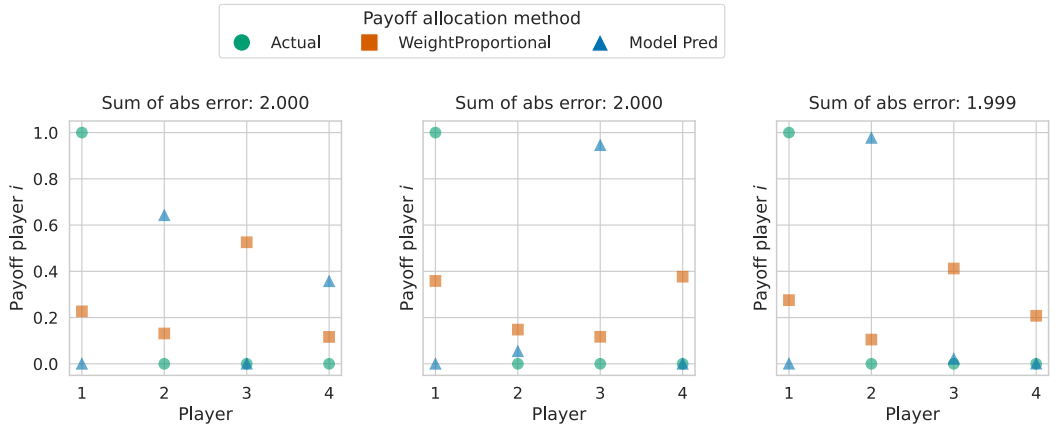

Figure I.2: **Examples of games where the joint payoff is allocated to a single player**. We depict the actual solutions (green dots), payoffs allocated according to the weight proportional heuristic (see eq. H, orange squares) and model predictions (blue triangles).

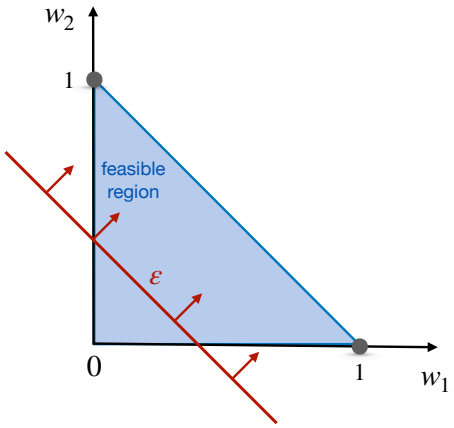

Figure I.3: **Visual depiction of the corner solutions and feasible region of a two-player weighted voting game**. Take the weighted voting game $\mathbf{w} = (1, 1)$ with $q = 2$. Simplex based methods tend to return corner solutions, in this case $\mathbf{p} = (1, 0)$ or $\mathbf{p} = (0, 1)$ with a LCV $= 0$.

J Applying Our Models to Voting In The European Council

In this section we apply our trained models to predict voting power of member states in the EU Council. First, we take the weights from four countries in the EU voting council [31], see Table 7, and define the quota as the majority vote (50 % of the total weights + 1). Our (weighted) voting game consists of the member states Hungary, Netherlands, Poland and Ireland with the weights $\mathbf{w} = (12, 13, 27, 7)$, and quota $q = 30, 5$.

Following, we divide the weights by the quota and feed the normalized weights (see Section 3.1). We contrast the model predictions against the ground-truth solutions and find that the Mean Mean Absolute Error (MMAE) between our model predictions and actual solutions per member state is 0.003 (Banzhaf), 0.002 (Shapley), 0.017 and 0.007 for the least core payoffs and excess respectively. Figure J.1 compared the payoff allocations for the Shapley, Banzhaf and Core solution concepts. We observe that the contribution measured by the Power indices (Shapley and Banzhaf) is less than the weight, while the Core attributes a larger share of the joint payoff to this member state.

Next, we perform a sensitivity analysis. In such an analysis, we are interested in the interdependence of member states' voting power. Holding all other game parameters constant, we increment the weight of Hungary ($w_{\text{Hungary}} = 7$) with values of one, until it exceeds the quota of 30,5. Figure J.2 depicts how the member states' voting power (in Shapley value) changes as Hungary gains

Table 7: Comparison of voting weights in the Council of the European Union [31].

Member state	Population	Population Perc.	Weights	Weights Perc.
Germany	82.54m	16.5%	29	8.4%
France	59.64m	12.9%	29	8.4%
UK	59.33m	12.4%	29	8.4%
Italy	57.32m	12.0%	29	8.4%
Spain	41.55m	9.0%	27	7.8%
Poland	38.22m	7.6%	27	7.8%
Romania	21.77m	4.3%	14	4.1%
Netherlands	17,02m	3.3%	13	3.8%
Greece	11.01m	2.2%	12	3.5%
Portugal	10.41m	2.1%	12	3.5%
Belgium	10.36m	2.1%	12	3.5%
Czech Rep.	10.20m	2.1%	12	3.5%
Hungary	10.14m	2.0%	12	3.5%
Sweden	8.94m	1.9%	10	2.9%
Austria	8.08m	1.7%	10	2.9%
Bulgaria	7.85m	1.5%	10	2.9%
Denmark	5.38m	1.1%	7	2.0%
Slovakia	5.38m	1.1%	7	2.0%
Finland	5.21m	1.1%	7	2.0%
Ireland	3.96m	0.9%	7	2.0%

absolute power (as we increase the weight). The predicted payoff for Hungary is depicted by the red triangles, the model predictions for the other states in blue, and the actual solutions by the green dots. We observe that there are two transition points in the solution space. One transition point occurs when $w_{\text{Hungary}} = 17.5/18.5$. Hungary can now form a winning coalition both with the Netherlands ($w_{\text{Netherlands}} = 12$) and with Poland ($w_{\text{Poland}} = 13$), which raises its power in the voting and accordingly its' share of the joint payoff. We notice that our model is able to predict accurate payoffs overall, with a spike in absolute error per player at this transition point.

The second transition point occurs when $w_{\text{Hungary}} = 30.5 = q$. Hungary is now also a winning coalition by itself, thereby obtaining a larger share of the joint payoff. We observe that its' Shapley value increases to approximately 0.6, obtaining the more than half of the total reward. Our model is able to capture this transition almost perfectly (MMAE < 0.01).

We perform a similar analysis, this time perturbing the quota. Figure J.3 depicts the same game with the original weights fixed, increasing the value of the quota from small to the sum of the weights (notice that this is the same experiment we performed in Section 4.1.2). The value of the quota determines the set of winning coalitions, thereby dictating the payoff of each player. As such, perturbing in this game yields a large number of discontinuous jumps in the member states solutions compared to our previous analysis perturbing the weight of Hungary. Our model is able to capture the overall changes in the solutions but has difficulty with capturing large jumps in payoffs for small quota increments.

Our variable models models have been trained on games up to ten players and can be applied to instantly predict the payoffs of member states in larger councils. To illustrate this, we take all member states in Table 7 and use our variable models to predict the payoffs for each, again setting the quota as the majority vote. We report the prediction quality of our models with respect to the

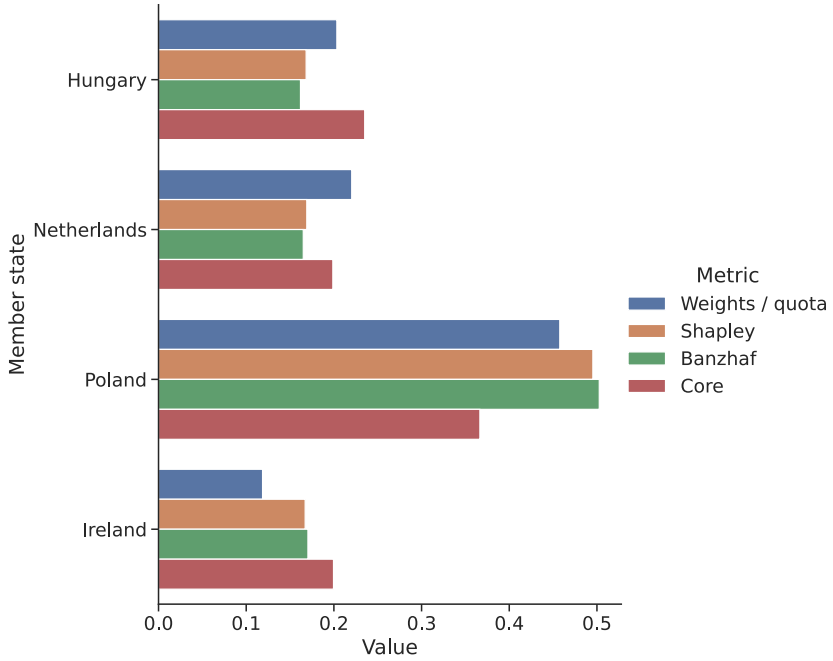

Figure J.1: **Allocating payoffs to member states in the EU Council.** A comparison of how absolute power (the normalized weights $(\mathbf{w}/q)$) relates to the allocated share of the joint payoff.

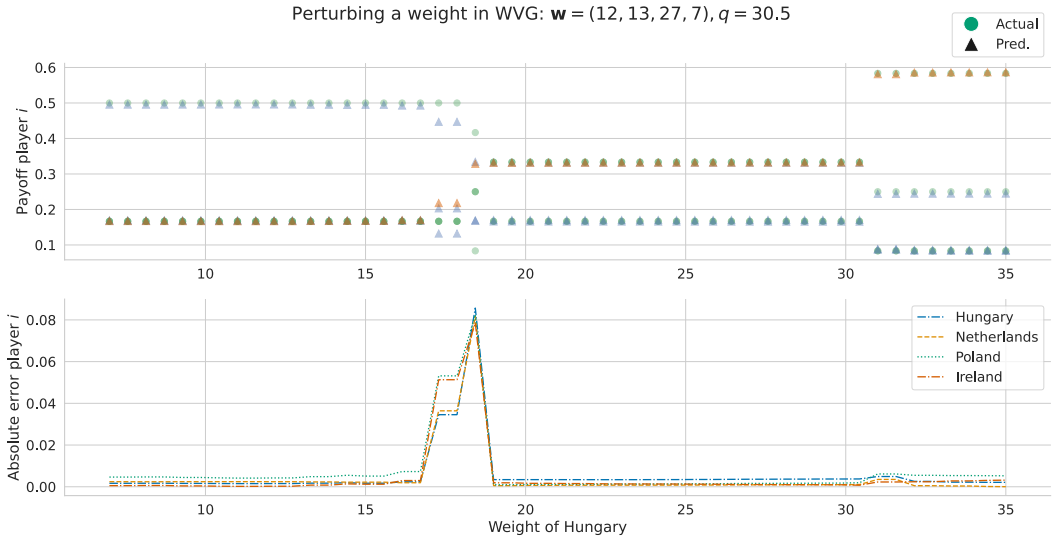

Figure J.2: **Perturbing the weight of a member state in a voting game.** *Top*: Model predictions and actual payoffs for a four-player voting game with weight vector $\mathbf{w} = (12, 13, 27, 7)$. textitBottom: Absolute errors between the individual member state payoffs.

ground truth (in Mean Mean Absolute Error), which is 0.013 for Shapley, 0.004 for Banzhaf, and 0.05 for the least core. Figure J.4 compares the solutions across member states and solution concepts.

While there is room for improvement in terms of prediction quality, our approach provides a useful tool for obtaining quick estimates of the voting power in medium size games ($10 \leq n < 50$). Furthermore, while approximation methods exist for Shapley and Banzhaf, there are currently no effective methods for estimating payoffs in the least core. This solution concept is more challenging as the solutions are obtained by solving a Linear Program with a hard constraints. Thus, solving a

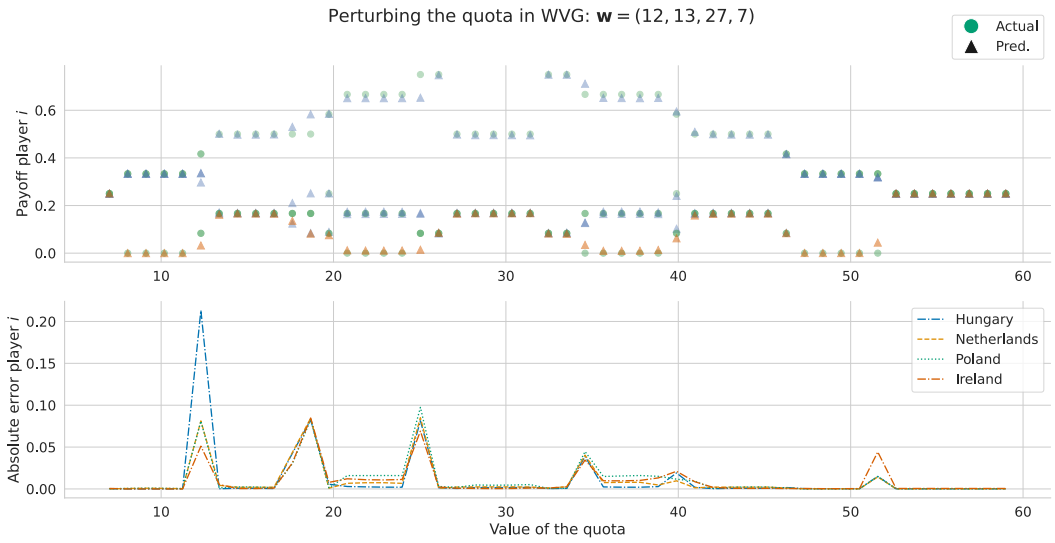

Figure J.3: **Perturbing the quota in a voting game.** *Top*: Model predictions and actual payoffs for a four-player voting game with weight vector $\mathbf{w} = (12, 13, 27, 7)$. *Bottom*: Absolute errors between the individual member state payoffs. Note that these are a large number of transition points in the solutions.

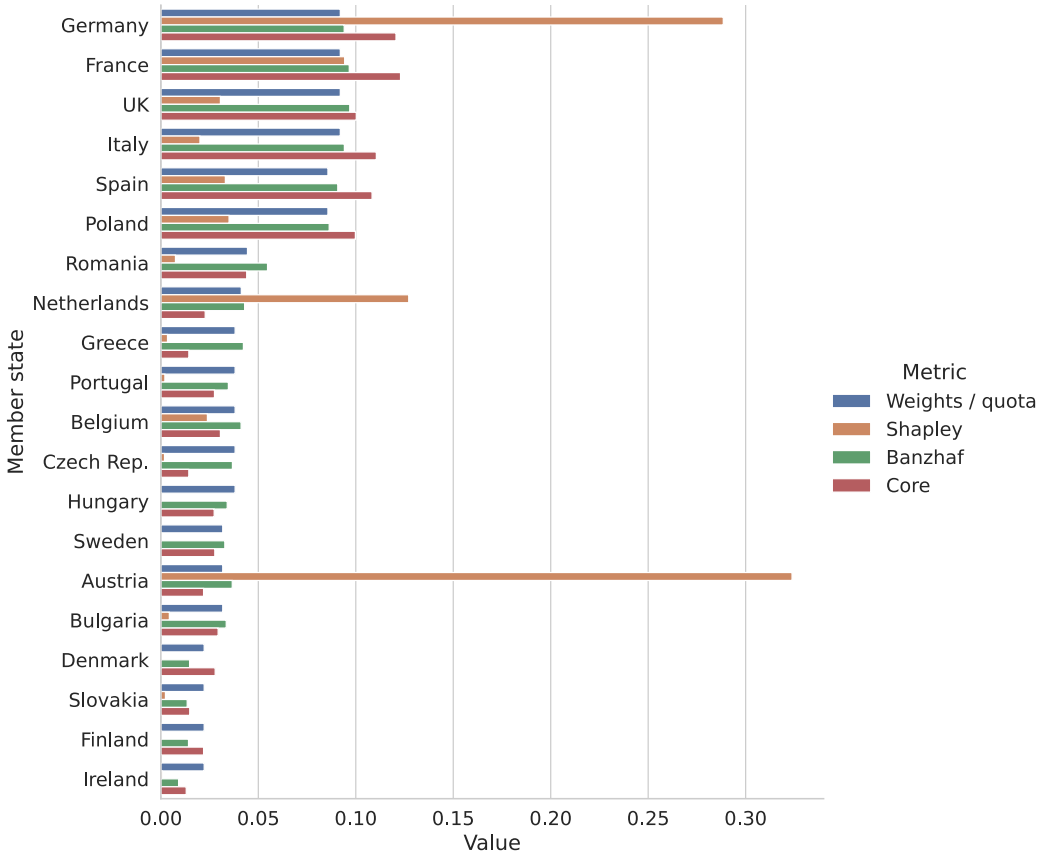

Figure J.4: **Allocating payoffs to member states in the EU council with variable models**. A comparison of the allocated payoffs over solution concepts.

game with 15 or even 20 players contains on the order of 2^{15} or 2^{20} constraints, which slow down the solvers dramatically. Here, we can apply the same technique, solve for the small/medium size-games, then leverage our neural network approach to tackle a higher numbers of players. In other words, provided a solver that can handle up to M players, we can train a neural network on this number of players, allowing us to extend the number of players even beyond M, possibly up to $2M$.

K Real World Applications of Weighted Voting Games

Weighted voting games, and cooperative game theory in general, provide valuable tools for reasoning about the allocation of collective outcomes to individual agents. As such, their applicability span numerous domains. In this section, we highlight several real-world applications of our work. In all examples, our framework can be applied to both speed up and enhance the scalability. Dinar et al. [18] utilize solution concepts from cooperative game theory – including the Core, the Shapley Value, the Nucleolus and Nash – to allocate water resources to competing parties in the presence of scarcity. More recently, Mirzaei-Nodoushan et al. [41] also performed an extensive evaluation of both cooperative and non-cooperative game theoretic approach for water management in trans-boundary river basins.

Moreover, cooperative game theory has proven useful in analyzing the costs of joint activities [57]. Vázquez-Brage et al. [53] combine a model of airport games with the Shapley value to fairly distribute the aircraft landing fees among airlines. When an airport is build, the size of the runway is chosen to match the largest designed airplaine. The authors address how to fairly divide costs between airlines provided the different designs and sizes of their planes.

Bistaffa et al. [11] applied cooperative game theory to analyze the social ridesharing problem. Mobility platforms such as Lyft allow their users to share their trip with nearby users in real-time, providing a more environmentally friendly and cost effective alternative to commuting alone. They study how the costs of the ride should be divided among passengers. Another application is in supply chain management, where Fiestras-Janeiro et al. [22] study how to coordinate actions between suppliers, manufactorers, retailers and customers.