**Supplementary Materials**

**The Prediction of Mechanical Strength of Alkali-Activated Lunar Regolith Simulant under Extreme Lunar Environments: An Interpretable Stacking Ensemble Model Integrated with Hyperparameter Optimization Algorithms**

Yizhou Yao 1; Wei Zhang 2; Chao Zhu 3; Chao Liu 4,\*

1 Ph.D. Candidate, College of Civil Engineering, Xi'an University of Architecture and Technology, Xi'an, 710055, China. ORCID: <https://orcid.org/0009-0001-4097-3988>. Email: yaoyizhou@xauat.edu.cn

2 Ph.D. Candidate, College of Civil Engineering, Xi'an University of Architecture and Technology, Xi'an, 710055, China. Email: 2310521@tongji.edu.cn

3 Lecturer, College of Civil Engineering, Xi'an University of Architecture and Technology, Xi'an, 710055, China. Email: chaozhu@xauat.edu.cn

4 Professor, College of Civil Engineering, Xi'an University of Architecture and Technology, Xi'an, 710055, China. (corresponding author) Email: chaoliu@xauat.edu.cn

**The files include:**

**Supplementary Figures**

Fig. S1

Fig. S2

Fig. S3

Fig. S4

Fig. S5

**Supplementary Tables**

Table S1

**Supplementary Codes**

Code S1 Stacking-RS model

Code S2 Stacking-BO model

Code S3 Stacking-PSO model

Code S4 Stacking-SA model

Code S5 Stacking-GWO model

Code S6 Stacking-TPE model

Code S7 GUI tool for AALRS strength prediction

***Interaction analysis***

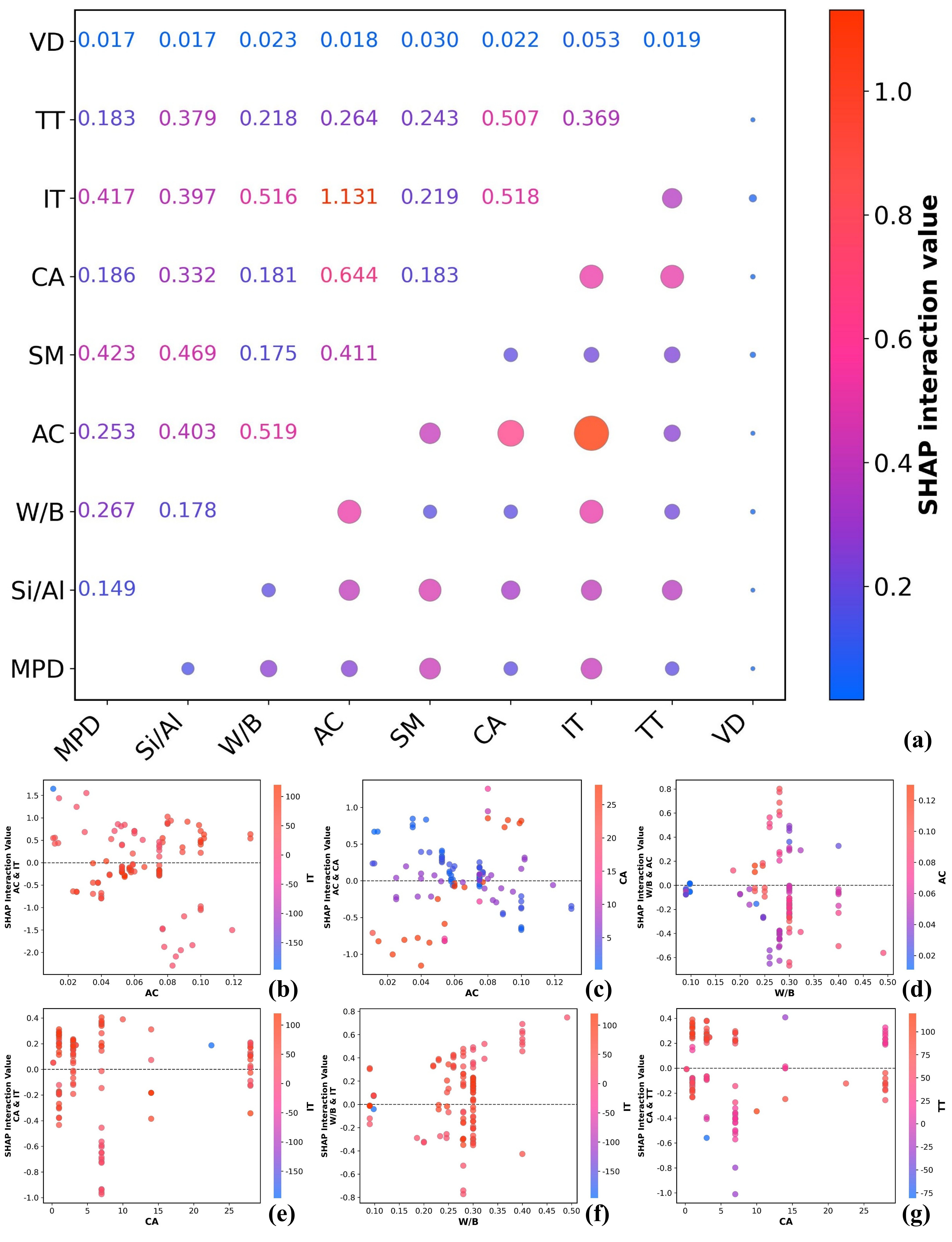


Fig. S1. SHAP interaction plots. (a) Summary plot of interaction value. (b) AC and IT, (c) AC and CA, (d) W/B and AC, (e) CA and IT, (f) W/B and IT, and (g) CA and TT.

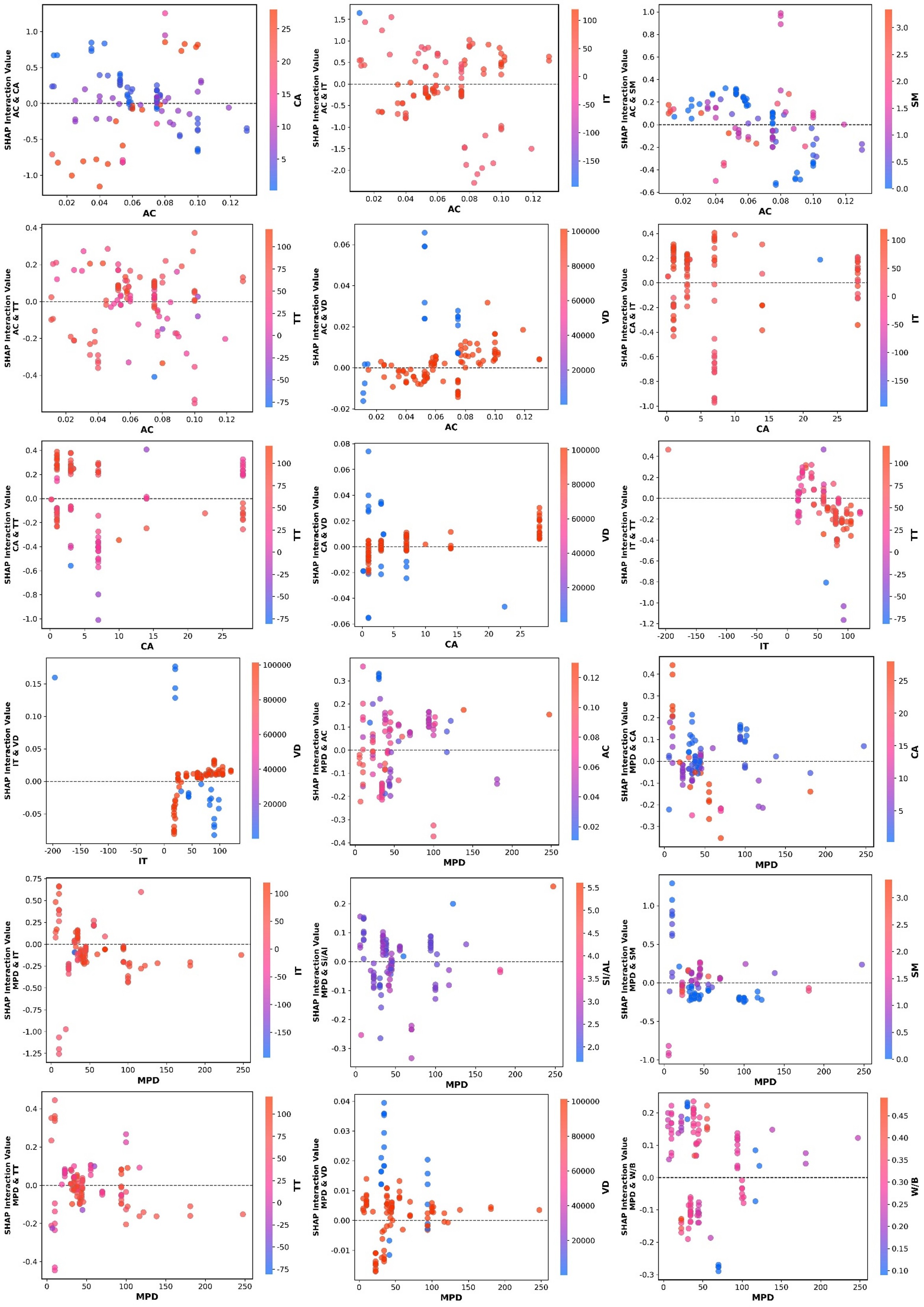


Fig. S2. SHAP interaction scatter plots (Interactions between AC, CA, IT, MPD and other features)

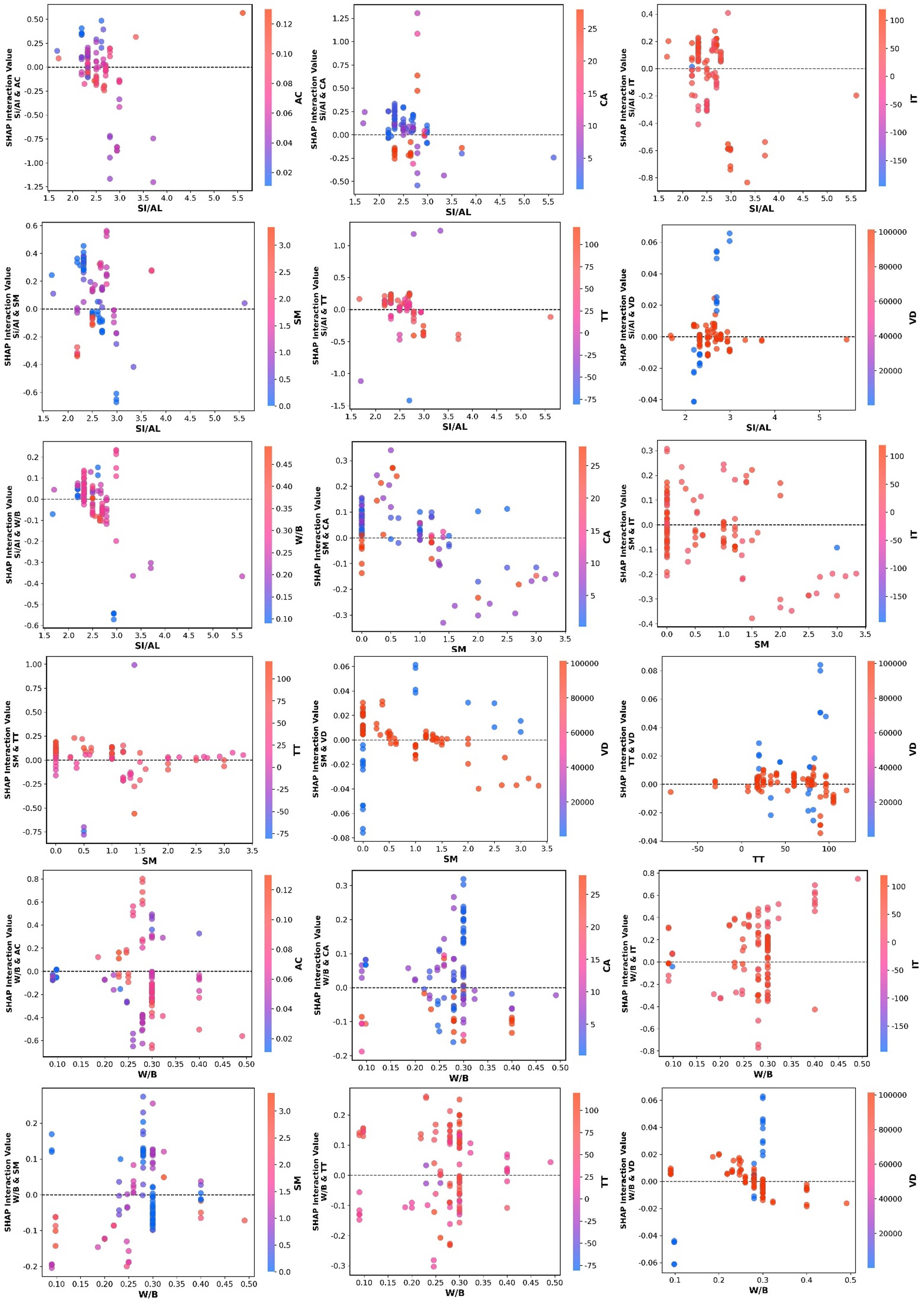


Fig. S3. SHAP interaction scatter plots (Interactions between Si/Al, SM, W/B and other features)

***2D PDP analysis***

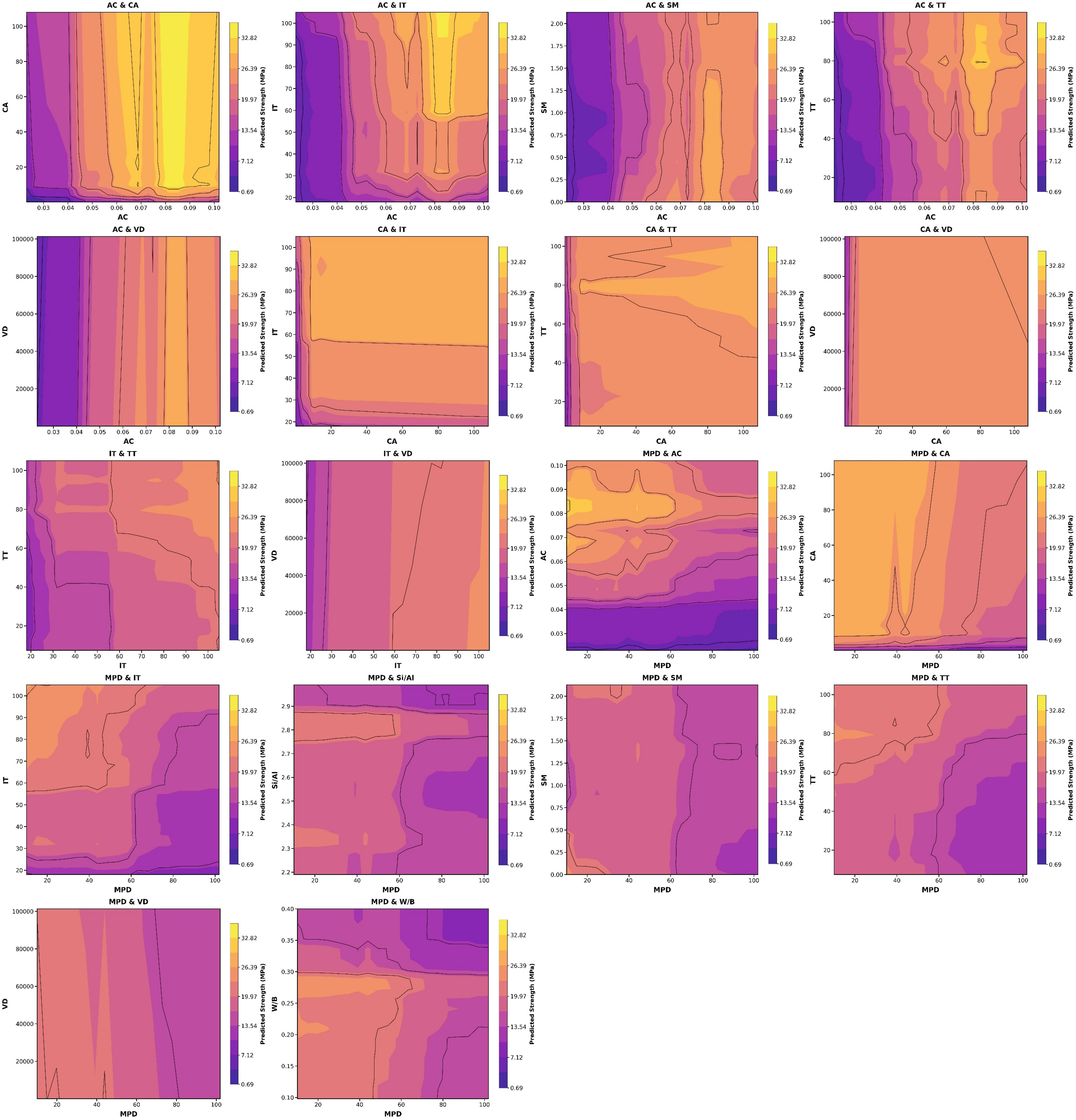


Fig. S4. 2D PDP analysis of the double features.

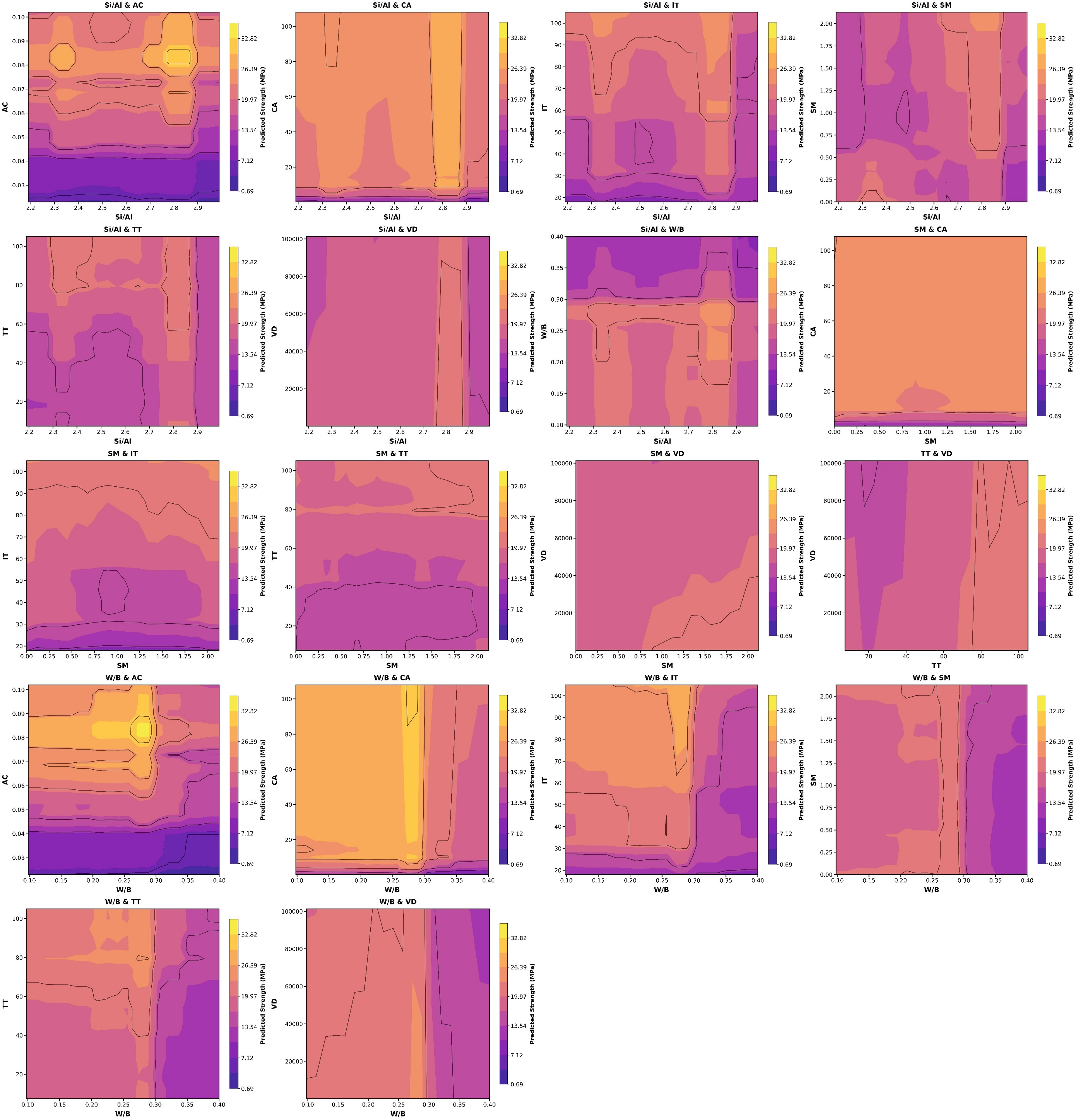


Fig. S5. 2D PDP analysis of the double features.

**Supplementary Tables**

**Table S1 Dataset of compressive strength of alkali-activated lunar regolith simulant from 24 classic studies and experimental results.**

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **MPD** | **Si/Al** | **W/B** | **AC** | **SM** | **CA** | **IT** | **TT** | **VD** | **CS** |
| 1 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 1 | 90 | 90 | 101325 | 11.96 |
| 2 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 3 | 90 | 90 | 101325 | 12.36 |
| 3 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 7 | 90 | 90 | 101325 | 13.2 |
| 4 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 14 | 90 | 90 | 101325 | 13.6 |
| 5 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 28 | 90 | 90 | 101325 | 14.2 |
| 6 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 1 | 90 | 90 | 101325 | 17.36 |
| 7 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 3 | 90 | 90 | 101325 | 18.63 |
| 8 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 7 | 90 | 90 | 101325 | 18.96 |
| 9 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 14 | 90 | 90 | 101325 | 19.2 |
| 10 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 28 | 90 | 90 | 101325 | 19.3 |
| 11 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 1 | 20 | 20 | 100 | 2.68 |
| 12 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 3 | 20 | 20 | 100 | 3.66 |
| 13 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 7 | 20 | 20 | 100 | 4.16 |
| 14 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 14 | 20 | 20 | 100 | 4.58 |
| 15 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 28 | 20 | 20 | 100 | 4.82 |
| 16 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 1 | 20 | 20 | 100 | 3.12 |
| 17 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 3 | 20 | 20 | 100 | 7.87 |
| 18 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 7 | 20 | 20 | 100 | 13.26 |
| 19 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 14 | 20 | 20 | 100 | 15.68 |
| 20 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 28 | 20 | 20 | 100 | 18.89 |
| 21 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 1 | 90 | 90 | 100 | 8.42 |
| 22 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 3 | 90 | 90 | 100 | 8.3 |
| 23 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 7 | 90 | 90 | 100 | 8.93 |
| 24 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 14 | 90 | 90 | 100 | 9.1 |
| 25 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 28 | 90 | 90 | 100 | 9.3 |
| 26 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 1 | 90 | 90 | 100 | 27.48 |
| 27 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 3 | 90 | 90 | 100 | 27.42 |
| 28 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 7 | 90 | 90 | 100 | 29.81 |
| 29 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 14 | 90 | 90 | 100 | 30.56 |
| 30 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 28 | 90 | 90 | 100 | 32.89 |
| 31 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 3 | 20 | 20 | 101325 | 0.15 |
| 32 | 34.35 | 2.7 | 0.3 | 0.075 | 0.5 | 3 | 20 | 20 | 101325 | 0.5 |
| 33 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 3 | 20 | 20 | 101325 | 0.33 |
| 34 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 3 | -81.73 | 65.16 | 101325 | 0.49 |
| 35 | 34.35 | 2.7 | 0.3 | 0.075 | 0.5 | 3 | -81.73 | 65.16 | 101325 | 0.47 |
| 36 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 3 | -81.73 | 65.16 | 101325 | 2.29 |
| 37 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 3 | 64.12 | -80.73 | 101325 | 1.02 |
| 38 | 34.35 | 2.7 | 0.3 | 0.075 | 0.5 | 3 | 64.12 | -80.73 | 101325 | 3.93 |
| 39 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 3 | 64.12 | -80.73 | 101325 | 8.68 |
| 40 | 34.35 | 2.7 | 0.3 | 0.075 | 0 | 3 | 62.7 | 98.8 | 101325 | 5.55 |
| 41 | 34.35 | 2.7 | 0.3 | 0.075 | 0.5 | 3 | 62.7 | 98.8 | 101325 | 12.46 |
| 42 | 34.35 | 2.7 | 0.3 | 0.075 | 1 | 3 | 62.7 | 98.8 | 101325 | 18.46 |
| 43 | 7 | 3.34 | 0.23 | 0.102 | 0.5 | 3 | 20 | 20 | 101325 | 8.62 |
| 44 | 7 | 3.34 | 0.23 | 0.102 | 0.5 | 3 | 93 | 93 | 101325 | 8.34 |
| 45 | 7 | 3.34 | 0.23 | 0.102 | 0.5 | 7 | 20 | 20 | 101325 | 13.04 |
| 46 | 7 | 3.34 | 0.23 | 0.102 | 0.5 | 7 | 93 | -30 | 101325 | 10.04 |
| 47 | 60 | 1.7 | 0.23 | 0.102 | 0.5 | 3 | 20 | 20 | 101325 | 1.24 |
| 48 | 60 | 1.7 | 0.23 | 0.102 | 0.5 | 3 | 93 | 93 | 101325 | 31.26 |
| 49 | 60 | 1.7 | 0.23 | 0.102 | 0.5 | 7 | 20 | 20 | 101325 | 1.53 |
| 50 | 60 | 1.7 | 0.23 | 0.102 | 0.5 | 7 | 93 | -30 | 101325 | 30.44 |
| 51 | 44.1 | 2.19 | 0.23 | 0.102 | 0.5 | 3 | 99.6 | 99.6 | 101325 | 29 |
| 52 | 33.5 | 2.52 | 0.23 | 0.102 | 0.5 | 3 | 99.6 | 99.6 | 101325 | 29.75 |
| 53 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 1 | 66.6 | 96.3 | 100 | 4.2 |
| 54 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 1 | 93 | 96.9 | 100 | 18.8 |
| 55 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 1 | 98.2 | 76.2 | 100 | 21.2 |
| 56 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 1 | 84.5 | 33.5 | 100 | 14.4 |
| 57 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 1 | 66.6 | 96.3 | 101325 | 5.9 |
| 58 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 1 | 93 | 96.9 | 101325 | 20.3 |
| 59 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 1 | 98.2 | 76.2 | 101325 | 23.2 |
| 60 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 1 | 84.5 | 33.5 | 101325 | 11.9 |
| 61 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 30.7 | 83.3 | 100 | 15.5 |
| 62 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 66.6 | 96.3 | 100 | 24.6 |
| 63 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 93 | 96.9 | 100 | 38.2 |
| 64 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 98.2 | 76.2 | 100 | 31.4 |
| 65 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 84.5 | 33.5 | 100 | 20.5 |
| 66 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 30.7 | 83.3 | 101325 | 7.7 |
| 67 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 66.6 | 96.3 | 101325 | 16.9 |
| 68 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 93 | 96.9 | 101325 | 33.2 |
| 69 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 98.2 | 76.2 | 101325 | 24.1 |
| 70 | 94.1 | 2.32 | 0.28 | 0.0525 | 0 | 3 | 84.5 | 33.5 | 101325 | 14.9 |
| 71 | 38.2 | 2.32 | 0.28 | 0.0698 | 0 | 1 | 60 | 60 | 101325 | 22 |
| 72 | 38.2 | 2.32 | 0.28 | 0.0698 | 0 | 3 | 60 | 60 | 101325 | 40.9 |
| 73 | 10.11 | 2.32 | 0.28 | 0.023 | 0 | 7 | 20 | 20 | 101325 | 0.8 |
| 74 | 10.11 | 2.32 | 0.28 | 0.05 | 0.53 | 7 | 20 | 20 | 101325 | 1.5 |
| 75 | 10.11 | 2.32 | 0.28 | 0.077 | 0.37 | 7 | 20 | 20 | 101325 | 4.7 |
| 76 | 10.11 | 2.32 | 0.28 | 0.104 | 0.31 | 7 | 20 | 20 | 101325 | 5.1 |
| 77 | 10.11 | 2.32 | 0.28 | 0.1 | 0.26 | 7 | 40 | 40 | 101325 | 35.1 |
| 78 | 10.11 | 2.32 | 0.28 | 0.073 | 0.23 | 7 | 40 | 40 | 101325 | 46.6 |
| 79 | 10.11 | 2.32 | 0.28 | 0.046 | 0 | 7 | 40 | 40 | 101325 | 43.7 |
| 80 | 10.11 | 2.32 | 0.28 | 0.071 | 0.85 | 7 | 40 | 40 | 101325 | 21.3 |
| 81 | 10.11 | 2.32 | 0.28 | 0.055 | 0.85 | 7 | 60 | 60 | 101325 | 4.3 |
| 82 | 10.11 | 2.32 | 0.28 | 0.082 | 0.47 | 7 | 60 | 60 | 101325 | 67.2 |
| 83 | 10.11 | 2.32 | 0.28 | 0.069 | 0 | 7 | 60 | 60 | 101325 | 74.3 |
| 84 | 10.11 | 2.32 | 0.28 | 0.096 | 0.14 | 7 | 60 | 60 | 101325 | 49.6 |
| 85 | 10.11 | 2.32 | 0.28 | 0.092 | 0 | 7 | 80 | 80 | 101325 | 62 |
| 86 | 10.11 | 2.32 | 0.28 | 0.099 | 0.33 | 7 | 80 | 80 | 101325 | 62 |
| 87 | 10.11 | 2.32 | 0.28 | 0.066 | 0.53 | 7 | 80 | 80 | 101325 | 60.2 |
| 88 | 10.11 | 2.32 | 0.28 | 0.027 | 0.85 | 7 | 80 | 80 | 101325 | 7.8 |
| 89 | 10.11 | 2.32 | 0.28 | 0.023 | 0 | 28 | 20 | 20 | 101325 | 1.1 |
| 90 | 10.11 | 2.32 | 0.28 | 0.05 | 0.53 | 28 | 20 | 20 | 101325 | 1.5 |
| 91 | 10.11 | 2.32 | 0.28 | 0.077 | 0.37 | 28 | 20 | 20 | 101325 | 7.5 |
| 92 | 10.11 | 2.32 | 0.28 | 0.104 | 0.31 | 28 | 20 | 20 | 101325 | 18.5 |
| 93 | 10.11 | 2.32 | 0.28 | 0.1 | 0.26 | 28 | 40 | 40 | 101325 | 41.7 |
| 94 | 10.11 | 2.32 | 0.28 | 0.073 | 0.23 | 28 | 40 | 40 | 101325 | 39.9 |
| 95 | 10.11 | 2.32 | 0.28 | 0.046 | 0 | 28 | 40 | 40 | 101325 | 39.3 |
| 96 | 10.11 | 2.32 | 0.28 | 0.071 | 0.85 | 28 | 40 | 40 | 101325 | 21.5 |
| 97 | 10.11 | 2.32 | 0.28 | 0.055 | 0.85 | 28 | 60 | 60 | 101325 | 5.5 |
| 98 | 10.11 | 2.32 | 0.28 | 0.082 | 0.47 | 28 | 60 | 60 | 101325 | 69.3 |
| 99 | 10.11 | 2.32 | 0.28 | 0.069 | 0 | 28 | 60 | 60 | 101325 | 69 |
| 100 | 10.11 | 2.32 | 0.28 | 0.096 | 0.14 | 28 | 60 | 60 | 101325 | 48.2 |
| 101 | 10.11 | 2.32 | 0.28 | 0.092 | 0 | 28 | 80 | 80 | 101325 | 59.5 |
| 102 | 10.11 | 2.32 | 0.28 | 0.099 | 0.33 | 28 | 80 | 80 | 101325 | 75.6 |
| 103 | 10.11 | 2.32 | 0.28 | 0.066 | 0.53 | 28 | 80 | 80 | 101325 | 63.7 |
| 104 | 10.11 | 2.32 | 0.28 | 0.027 | 0.85 | 28 | 80 | 80 | 101325 | 14.3 |
| 105 | 45 | 2.61 | 0.21 | 0.0481 | 1.14 | 3 | 60 | 60 | 101325 | 16.62 |
| 106 | 45 | 2.61 | 0.21 | 0.0481 | 1.14 | 7 | 23 | 23 | 101325 | 12.05 |
| 107 | 19 | 2.61 | 0.233 | 0.0145 | 0 | 28 | 20 | 20 | 101325 | 2 |
| 108 | 19 | 2.61 | 0.233 | 0.029 | 0 | 28 | 20 | 20 | 101325 | 3.7 |
| 109 | 19 | 2.61 | 0.233 | 0.043 | 0 | 28 | 20 | 20 | 101325 | 7.8 |
| 110 | 19 | 2.61 | 0.233 | 0.058 | 0 | 28 | 20 | 20 | 101325 | 18.4 |
| 111 | 68.2 | 2.61 | 0.219 | 0.045 | 2 | 3 | 80 | 80 | 101325 | 19.78 |
| 112 | 68.2 | 2.61 | 0.219 | 0.045 | 2 | 7 | 80 | 80 | 101325 | 21.52 |
| 113 | 68.2 | 2.61 | 0.219 | 0.045 | 2 | 28 | 80 | 80 | 101325 | 23.48 |
| 114 | 35.3 | 2.65 | 0.219 | 0.045 | 2 | 3 | 80 | 80 | 101325 | 39.89 |
| 115 | 35.3 | 2.65 | 0.219 | 0.045 | 2 | 7 | 80 | 80 | 101325 | 47.83 |
| 116 | 35.3 | 2.65 | 0.219 | 0.045 | 2 | 28 | 80 | 80 | 101325 | 51.96 |
| 117 | 58 | 2.54 | 0.219 | 0.045 | 2 | 3 | 80 | 80 | 101325 | 18.91 |
| 118 | 58 | 2.54 | 0.219 | 0.045 | 2 | 7 | 80 | 80 | 101325 | 21.3 |
| 119 | 58 | 2.54 | 0.219 | 0.045 | 2 | 28 | 80 | 80 | 101325 | 21.52 |
| 120 | 46.2 | 1.7 | 0.219 | 0.045 | 2 | 3 | 80 | 80 | 101325 | 7.93 |
| 121 | 46.2 | 1.7 | 0.219 | 0.045 | 2 | 7 | 80 | 80 | 101325 | 16.74 |
| 122 | 46.2 | 1.7 | 0.219 | 0.045 | 2 | 28 | 80 | 80 | 101325 | 20.87 |
| 123 | 181 | 3.71 | 0.219 | 0.045 | 2 | 3 | 80 | 80 | 101325 | 0.87 |
| 124 | 181 | 3.71 | 0.219 | 0.045 | 2 | 7 | 80 | 80 | 101325 | 3.15 |
| 125 | 181 | 3.71 | 0.219 | 0.045 | 2 | 28 | 80 | 80 | 101325 | 17.83 |
| 126 | 22.7 | 2.5 | 0.491 | 0.155 | 0.85 | 7 | 18 | 18 | 101325 | 0.19 |
| 127 | 22.7 | 2.5 | 0.4 | 0.129 | 0.79 | 7 | 18 | 18 | 101325 | 1.09 |
| 128 | 22.7 | 2.5 | 0.323 | 0.107 | 0.74 | 7 | 18 | 18 | 101325 | 2.39 |
| 129 | 22.7 | 2.5 | 0.257 | 0.077 | 0.96 | 7 | 18 | 18 | 101325 | 3.02 |
| 130 | 22.7 | 2.5 | 0.2 | 0.062 | 0.88 | 7 | 18 | 18 | 101325 | 5.3 |
| 131 | 22.7 | 2.5 | 0.491 | 0.129 | 1.33 | 7 | 18 | 18 | 101325 | 0.31 |
| 132 | 22.7 | 2.5 | 0.4 | 0.103 | 1.33 | 7 | 18 | 18 | 101325 | 0.53 |
| 133 | 22.7 | 2.5 | 0.323 | 0.083 | 1.34 | 7 | 18 | 18 | 101325 | 0.77 |
| 134 | 22.7 | 2.5 | 0.257 | 0.066 | 1.32 | 7 | 18 | 18 | 101325 | 1.3 |
| 135 | 22.7 | 2.5 | 0.2 | 0.052 | 1.33 | 7 | 18 | 18 | 101325 | 1.39 |
| 136 | 22.7 | 2.5 | 0.491 | 0.1 | 2.11 | 7 | 18 | 18 | 101325 | 0.78 |
| 137 | 22.7 | 2.5 | 0.4 | 0.078 | 2.2 | 7 | 18 | 18 | 101325 | 0.87 |
| 138 | 22.7 | 2.5 | 0.323 | 0.071 | 1.76 | 7 | 18 | 18 | 101325 | 1.41 |
| 139 | 22.7 | 2.5 | 0.257 | 0.055 | 1.86 | 7 | 18 | 18 | 101325 | 2.7 |
| 140 | 22.7 | 2.5 | 0.2 | 0.041 | 2 | 7 | 18 | 18 | 101325 | 4.61 |
| 141 | 22.7 | 2.5 | 0.491 | 0.085 | 2.64 | 7 | 18 | 18 | 101325 | 0.67 |
| 142 | 22.7 | 2.5 | 0.4 | 0.065 | 2.92 | 7 | 18 | 18 | 101325 | 0.72 |
| 143 | 22.7 | 2.5 | 0.323 | 0.048 | 3.34 | 7 | 18 | 18 | 101325 | 1.74 |
| 144 | 22.7 | 2.5 | 0.257 | 0.044 | 2.66 | 7 | 18 | 18 | 101325 | 3.63 |
| 145 | 22.7 | 2.5 | 0.2 | 0.031 | 3.15 | 7 | 18 | 18 | 101325 | 4.5 |
| 146 | 22.7 | 2.5 | 0.246 | 0.119 | 1.32 | 7 | 18 | 18 | 101325 | 0.03 |
| 147 | 22.7 | 2.5 | 0.186 | 0.088 | 1.33 | 7 | 18 | 18 | 101325 | 1.55 |
| 148 | 22.7 | 2.5 | 0.147 | 0.072 | 1.34 | 7 | 18 | 18 | 101325 | 3.59 |
| 149 | 22.7 | 2.5 | 0.364 | 0.142 | 2 | 7 | 18 | 18 | 101325 | 0.74 |
| 150 | 22.7 | 2.5 | 0.3 | 0.116 | 1.92 | 7 | 18 | 18 | 101325 | 1.96 |
| 151 | 22.7 | 2.5 | 0.246 | 0.095 | 2 | 7 | 18 | 18 | 101325 | 2.62 |
| 152 | 22.7 | 2.5 | 0.2 | 0.078 | 1.92 | 7 | 18 | 18 | 101325 | 4.01 |
| 153 | 22.7 | 2.5 | 0.147 | 0.052 | 2.13 | 7 | 18 | 18 | 101325 | 4.4 |
| 154 | 45 | 2.79 | 0.26 | 0.04 | 1.2 | 3 | 60 | 60 | 101325 | 8.96 |
| 155 | 45 | 2.79 | 0.28 | 0.06 | 1.2 | 3 | 60 | 60 | 101325 | 25.13 |
| 156 | 45 | 2.79 | 0.3 | 0.08 | 1.2 | 3 | 60 | 60 | 101325 | 40.43 |
| 157 | 45 | 2.79 | 0.28 | 0.04 | 1.4 | 3 | 60 | 60 | 101325 | 7.24 |
| 158 | 45 | 2.79 | 0.3 | 0.06 | 1.4 | 3 | 60 | 60 | 101325 | 20.29 |
| 159 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 3 | 60 | 60 | 101325 | 42.71 |
| 160 | 45 | 2.79 | 0.3 | 0.04 | 1.6 | 3 | 60 | 60 | 101325 | 5.39 |
| 161 | 45 | 2.79 | 0.26 | 0.06 | 1.6 | 3 | 60 | 60 | 101325 | 31.06 |
| 162 | 45 | 2.79 | 0.28 | 0.08 | 1.6 | 3 | 60 | 60 | 101325 | 40.61 |
| 163 | 45 | 2.79 | 0.26 | 0.04 | 1.2 | 7 | 60 | 60 | 101325 | 12.4 |
| 164 | 45 | 2.79 | 0.28 | 0.06 | 1.2 | 7 | 60 | 60 | 101325 | 44.4 |
| 165 | 45 | 2.79 | 0.3 | 0.08 | 1.2 | 7 | 60 | 60 | 101325 | 48.9 |
| 166 | 45 | 2.79 | 0.28 | 0.04 | 1.4 | 7 | 60 | 60 | 101325 | 13.9 |
| 167 | 45 | 2.79 | 0.3 | 0.06 | 1.4 | 7 | 60 | 60 | 101325 | 33.4 |
| 168 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 7 | 60 | 60 | 101325 | 58.8 |
| 169 | 45 | 2.79 | 0.3 | 0.04 | 1.6 | 7 | 60 | 60 | 101325 | 8.3 |
| 170 | 45 | 2.79 | 0.26 | 0.06 | 1.6 | 7 | 60 | 60 | 101325 | 37.4 |
| 171 | 45 | 2.79 | 0.28 | 0.08 | 1.6 | 7 | 60 | 60 | 101325 | 44.9 |
| 172 | 45 | 2.79 | 0.26 | 0.04 | 1.2 | 28 | 60 | 60 | 101325 | 16.8 |
| 173 | 45 | 2.79 | 0.28 | 0.06 | 1.2 | 28 | 60 | 60 | 101325 | 53.36 |
| 174 | 45 | 2.79 | 0.3 | 0.08 | 1.2 | 28 | 60 | 60 | 101325 | 56.91 |
| 175 | 45 | 2.79 | 0.28 | 0.04 | 1.4 | 28 | 60 | 60 | 101325 | 15.83 |
| 176 | 45 | 2.79 | 0.3 | 0.06 | 1.4 | 28 | 60 | 60 | 101325 | 41.63 |
| 177 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 28 | 60 | 60 | 101325 | 57.94 |
| 178 | 45 | 2.79 | 0.3 | 0.04 | 1.6 | 28 | 60 | 60 | 101325 | 11.74 |
| 179 | 45 | 2.79 | 0.26 | 0.06 | 1.6 | 28 | 60 | 60 | 101325 | 47.78 |
| 180 | 45 | 2.79 | 0.28 | 0.08 | 1.6 | 28 | 60 | 60 | 101325 | 51.2 |
| 181 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 8 | 60 | 120 | 101325 | 52.85 |
| 182 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 10 | 60 | 120 | 101325 | 50 |
| 183 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 14 | 60 | 120 | 101325 | 52.85 |
| 184 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 8 | 60 | 60 | 101325 | 59.15 |
| 185 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 10 | 60 | 60 | 101325 | 57.68 |
| 186 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 14 | 60 | 60 | 101325 | 58.55 |
| 187 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 8 | 60 | 0 | 101325 | 56.3 |
| 188 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 10 | 60 | 0 | 101325 | 80.73 |
| 189 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 14 | 60 | 0 | 101325 | 65.8 |
| 190 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 8 | 60 | -30 | 101325 | 65.45 |
| 191 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 10 | 60 | -30 | 101325 | 80.82 |
| 192 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 14 | 60 | -30 | 101325 | 81.5 |
| 193 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 8 | 60 | -196 | 101325 | 60.03 |
| 194 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 9 | 60 | -196 | 101325 | 68.52 |
| 195 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 10 | 60 | -196 | 101325 | 67.93 |
| 196 | 45 | 2.79 | 0.26 | 0.08 | 1.4 | 11 | 60 | -196 | 101325 | 67.85 |
| 197 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 1 | 66.6 | 74.9 | 101325 | 23.5 |
| 198 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 1 | 93 | 96.2 | 101325 | 29.3 |
| 199 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 1 | 98.2 | 96.9 | 101325 | 30.4 |
| 200 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 1 | 84.5 | 70.2 | 101325 | 26.7 |
| 201 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 3 | 30.7 | 83.3 | 101325 | 35.3 |
| 202 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 3 | 66.6 | 96.3 | 101325 | 37.7 |
| 203 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 3 | 93 | 99.6 | 101325 | 32.7 |
| 204 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 3 | 98.2 | 76.2 | 101325 | 31.2 |
| 205 | 38.2 | 2.32 | 0.3 | 0.057 | 0 | 3 | 84.5 | 33.5 | 101325 | 29.6 |
| 206 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 1 | 66.6 | 74.9 | 101325 | 25.4 |
| 207 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 1 | 93 | 96.2 | 101325 | 33.1 |
| 208 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 1 | 98.2 | 96.9 | 101325 | 33.9 |
| 209 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 1 | 84.5 | 70.2 | 101325 | 29.1 |
| 210 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 3 | 30.7 | 83.3 | 101325 | 35.8 |
| 211 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 3 | 66.6 | 96.3 | 101325 | 38.2 |
| 212 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 3 | 93 | 99.6 | 101325 | 36.8 |
| 213 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 3 | 98.2 | 76.2 | 101325 | 36.1 |
| 214 | 38.2 | 2.32 | 0.3 | 0.077 | 0 | 3 | 84.5 | 33.5 | 101325 | 31.2 |
| 215 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 1 | 66.6 | 74.9 | 101325 | 16 |
| 216 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 1 | 93 | 96.2 | 101325 | 26.6 |
| 217 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 1 | 98.2 | 96.9 | 101325 | 24.8 |
| 218 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 1 | 84.5 | 70.2 | 101325 | 17.3 |
| 219 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 3 | 30.7 | 83.3 | 101325 | 23.4 |
| 220 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 3 | 66.6 | 96.3 | 101325 | 27.4 |
| 221 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 3 | 93 | 99.6 | 101325 | 32.4 |
| 222 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 3 | 98.2 | 76.2 | 101325 | 30.1 |
| 223 | 38.2 | 2.32 | 0.3 | 0.1 | 0 | 3 | 84.5 | 33.5 | 101325 | 22.4 |
| 224 | 44 | 2.19 | 0.3 | 0.058 | 0 | 1 | 60 | 60 | 101325 | 1.54 |
| 225 | 44 | 2.19 | 0.3 | 0.058 | 0 | 1 | 82 | 82 | 101325 | 12.85 |
| 226 | 44 | 2.19 | 0.3 | 0.058 | 0 | 1 | 105 | 105 | 101325 | 28.26 |
| 227 | 44 | 2.19 | 0.279 | 0.043 | 0.5 | 1 | 82 | 82 | 101325 | 4.59 |
| 228 | 44 | 2.19 | 0.279 | 0.043 | 0.5 | 1 | 105 | 105 | 101325 | 9.84 |
| 229 | 44 | 2.19 | 0.247 | 0.035 | 1 | 1 | 60 | 60 | 101325 | 0.68 |
| 230 | 44 | 2.19 | 0.247 | 0.035 | 1 | 1 | 82 | 82 | 101325 | 2.38 |
| 231 | 44 | 2.19 | 0.247 | 0.035 | 1 | 1 | 105 | 105 | 101325 | 4.95 |
| 232 | 32 | 2.32 | 0.3 | 0.058 | 0 | 1 | 105 | 105 | 101325 | 14.27 |
| 233 | 32 | 2.32 | 0.279 | 0.043 | 0.5 | 1 | 82 | 82 | 101325 | 0.16 |
| 234 | 32 | 2.32 | 0.279 | 0.043 | 0.5 | 1 | 105 | 105 | 101325 | 23.9 |
| 235 | 32 | 2.32 | 0.247 | 0.035 | 1 | 1 | 82 | 82 | 101325 | 0.18 |
| 236 | 32 | 2.32 | 0.247 | 0.035 | 1 | 1 | 105 | 105 | 101325 | 23.14 |
| 237 | 43 | 2.99 | 0.3 | 0.058 | 0 | 1 | 82 | 82 | 101325 | 0.08 |
| 238 | 43 | 2.99 | 0.3 | 0.058 | 0 | 1 | 105 | 105 | 101325 | 0.18 |
| 239 | 43 | 2.99 | 0.279 | 0.043 | 0.5 | 1 | 82 | 82 | 101325 | 0.12 |
| 240 | 43 | 2.99 | 0.279 | 0.043 | 0.5 | 1 | 105 | 105 | 101325 | 13.74 |
| 241 | 43 | 2.99 | 0.247 | 0.035 | 1 | 1 | 82 | 82 | 101325 | 0.19 |
| 242 | 43 | 2.99 | 0.247 | 0.035 | 1 | 1 | 105 | 105 | 101325 | 14.71 |
| 243 | 156 | 2.32 | 0.3 | 0.058 | 0 | 1 | 105 | 105 | 101325 | 23.4 |
| 244 | 156 | 2.32 | 0.247 | 0.035 | 1 | 1 | 105 | 105 | 101325 | 29.05 |
| 245 | 214 | 2.99 | 0.247 | 0.035 | 1 | 1 | 105 | 105 | 101325 | 11.14 |
| 246 | 30.4 | 2.19 | 0.098 | 0.014 | 2 | 0.1875 | 44 | 44 | 10 | 16.12 |
| 247 | 30.4 | 2.19 | 0.098 | 0.012 | 2.5 | 0.1875 | 44 | 44 | 10 | 18.07 |
| 248 | 30.4 | 2.19 | 0.098 | 0.011 | 3 | 0.1875 | 44 | 44 | 10 | 13.61 |
| 249 | 30.4 | 2.19 | 0.098 | 0.014 | 2 | 3.375 | 44 | 78 | 10 | 16.24 |
| 250 | 30.4 | 2.19 | 0.098 | 0.012 | 2.5 | 3.375 | 44 | 78 | 10 | 17.75 |
| 251 | 30.4 | 2.19 | 0.098 | 0.011 | 3 | 3.375 | 44 | 78 | 10 | 15.04 |
| 252 | 30.4 | 2.19 | 0.098 | 0.014 | 2 | 22.5 | -196 | 78 | 10 | 19.3 |
| 253 | 30.4 | 2.19 | 0.098 | 0.012 | 2.5 | 22.5 | -196 | 78 | 10 | 20.04 |
| 254 | 30.4 | 2.19 | 0.098 | 0.011 | 3 | 22.5 | -196 | 78 | 10 | 17.48 |
| 255 | 30.4 | 2.19 | 0.098 | 0.014 | 2 | 108 | -196 | 78 | 10 | 15.81 |
| 256 | 30.4 | 2.19 | 0.098 | 0.012 | 2.5 | 108 | -196 | 78 | 10 | 17.17 |
| 257 | 30.4 | 2.19 | 0.098 | 0.011 | 3 | 108 | -196 | 78 | 10 | 15.04 |
| 258 | 30.4 | 2.19 | 0.3 | 0.075 | 0 | 1 | 82 | 82 | 101325 | 12.85 |
| 259 | 30.4 | 2.19 | 0.3 | 0.075 | 0 | 1 | 105 | 105 | 101325 | 28.26 |
| 260 | 30.4 | 2.19 | 0.3 | 0.075 | 0 | 1 | 82 | 82 | 10 | 0.49 |
| 261 | 30.4 | 2.19 | 0.3 | 0.075 | 1 | 1 | 82 | 82 | 101325 | 2.38 |
| 262 | 30.4 | 2.19 | 0.3 | 0.075 | 1 | 1 | 105 | 105 | 101325 | 4.95 |
| 263 | 30.4 | 2.19 | 0.3 | 0.075 | 1 | 1 | 82 | 82 | 10 | 13.28 |
| 264 | 101.9 | 2.32 | 0.3 | 0.075 | 0 | 1 | 82 | 82 | 101325 | 0.1 |
| 265 | 101.9 | 2.32 | 0.3 | 0.075 | 0 | 1 | 105 | 105 | 101325 | 14.27 |
| 266 | 101.9 | 2.32 | 0.3 | 0.075 | 0 | 1 | 82 | 82 | 10 | 0.63 |
| 267 | 101.9 | 2.32 | 0.3 | 0.075 | 1 | 1 | 82 | 82 | 101325 | 0.18 |
| 268 | 101.9 | 2.32 | 0.3 | 0.075 | 1 | 1 | 105 | 105 | 101325 | 23.14 |
| 269 | 101.9 | 2.32 | 0.3 | 0.075 | 1 | 1 | 82 | 82 | 10 | 15.5 |
| 270 | 41.6 | 2.99 | 0.3 | 0.075 | 0 | 1 | 82 | 82 | 101325 | 0.08 |
| 271 | 41.6 | 2.99 | 0.3 | 0.075 | 0 | 1 | 105 | 105 | 101325 | 0.18 |
| 272 | 41.6 | 2.99 | 0.3 | 0.075 | 0 | 1 | 82 | 82 | 10 | 0.38 |
| 273 | 41.6 | 2.99 | 0.3 | 0.075 | 1 | 1 | 82 | 82 | 101325 | 0.19 |
| 274 | 41.6 | 2.99 | 0.3 | 0.075 | 1 | 1 | 105 | 105 | 101325 | 14.71 |
| 275 | 41.6 | 2.99 | 0.3 | 0.075 | 1 | 1 | 82 | 82 | 10 | 12.16 |
| 276 | 39.45 | 2.55 | 0.4 | 0.06 | 1.2 | 3 | 60 | 60 | 101325 | 2.23 |
| 277 | 39.45 | 2.55 | 0.4 | 0.06 | 1.2 | 7 | 60 | 60 | 101325 | 4.69 |
| 278 | 39.45 | 2.55 | 0.4 | 0.06 | 1.2 | 28 | 60 | 60 | 101325 | 9.26 |
| 279 | 55.45 | 2.65 | 0.4 | 0.01 | 0 | 28 | 25 | 25 | 101325 | 10.66 |
| 280 | 55.45 | 2.65 | 0.4 | 0.02 | 0 | 28 | 25 | 25 | 101325 | 14.42 |
| 281 | 55.45 | 2.65 | 0.4 | 0.03 | 0 | 28 | 25 | 25 | 101325 | 16.49 |
| 282 | 55.45 | 2.65 | 0.4 | 0.04 | 0 | 28 | 25 | 25 | 101325 | 18.23 |
| 283 | 55.45 | 2.65 | 0.4 | 0.05 | 0 | 28 | 25 | 25 | 101325 | 18.53 |
| 284 | 55.45 | 2.65 | 0.4 | 0.06 | 0 | 28 | 25 | 25 | 101325 | 18.75 |
| 285 | 55.45 | 2.65 | 0.4 | 0.07 | 0 | 28 | 25 | 25 | 101325 | 17.95 |
| 286 | 55.45 | 2.65 | 0.4 | 0.08 | 0 | 28 | 25 | 25 | 101325 | 18.42 |
| 287 | 55.45 | 2.65 | 0.4 | 0.09 | 0 | 28 | 25 | 25 | 101325 | 17.03 |
| 288 | 55.45 | 2.65 | 0.4 | 0.1 | 0 | 28 | 25 | 25 | 101325 | 17.42 |
| 289 | 55.45 | 2.65 | 0.4 | 0.06 | 0.3 | 28 | 25 | 25 | 101325 | 19.74 |
| 290 | 55.45 | 2.65 | 0.4 | 0.06 | 0.6 | 28 | 25 | 25 | 101325 | 20.81 |
| 291 | 55.45 | 2.65 | 0.4 | 0.06 | 0.9 | 28 | 25 | 25 | 101325 | 24.88 |
| 292 | 55.45 | 2.65 | 0.4 | 0.06 | 1.2 | 28 | 25 | 25 | 101325 | 27.17 |
| 293 | 55.45 | 2.65 | 0.4 | 0.06 | 1.5 | 28 | 25 | 25 | 101325 | 26.69 |
| 294 | 55.45 | 2.65 | 0.4 | 0.06 | 1.8 | 28 | 25 | 25 | 101325 | 25.97 |
| 295 | 55.45 | 2.65 | 0.4 | 0.06 | 2.1 | 28 | 25 | 25 | 101325 | 24.6 |
| 296 | 55.45 | 2.65 | 0.4 | 0.06 | 2.4 | 28 | 25 | 25 | 101325 | 23.11 |
| 297 | 55.45 | 2.65 | 0.4 | 0.06 | 2.7 | 28 | 25 | 25 | 101325 | 24.73 |
| 298 | 55.45 | 2.65 | 0.4 | 0.06 | 3 | 28 | 25 | 25 | 101325 | 22.35 |
| 299 | 55.45 | 2.65 | 0.4 | 0.06 | 3.3 | 28 | 25 | 25 | 101325 | 22.1 |
| 300 | 117 | 2.61 | 0.09 | 0.0251 | 0 | 7 | 23 | 23 | 101325 | 0.06 |
| 301 | 117 | 2.61 | 0.09 | 0.0251 | 0 | 7 | 80 | 80 | 101325 | 1.76 |
| 302 | 122 | 1.67 | 0.09 | 0.0251 | 0 | 7 | 23 | 23 | 101325 | 0.63 |
| 303 | 122 | 1.67 | 0.09 | 0.0251 | 0 | 7 | 80 | 80 | 101325 | 1.7 |
| 304 | 70 | 2.94 | 0.09 | 0.0541 | 1.2 | 14 | 25 | 25 | 101325 | 1.02 |
| 305 | 70 | 2.94 | 0.09 | 0.0541 | 1.2 | 14 | 60 | 25 | 101325 | 1.53 |
| 306 | 70 | 2.94 | 0.09 | 0.0541 | 1.2 | 14 | 90 | 25 | 101325 | 4.28 |
| 307 | 70 | 2.94 | 0.09 | 0.0541 | 1.2 | 14 | 120 | 25 | 101325 | 8.16 |
| 308 | 70 | 2.94 | 0.09 | 0.0541 | 1.2 | 3 | 120 | 25 | 101325 | 7.88 |
| 309 | 70 | 2.94 | 0.09 | 0.0541 | 1.2 | 7 | 120 | 25 | 101325 | 9.02 |
| 310 | 70 | 2.94 | 0.09 | 0.0541 | 1.2 | 14 | 120 | 25 | 101325 | 10.67 |
| 311 | 70 | 2.94 | 0.09 | 0.0541 | 1.2 | 28 | 120 | 25 | 101325 | 13.16 |
| 312 | 5.76 | 2.67 | 0.25 | 0.1 | 1.5 | 1 | 60 | 60 | 101325 | 17.38 |
| 313 | 5.76 | 2.67 | 0.25 | 0.1 | 1.5 | 1 | 80 | 80 | 101325 | 27.93 |
| 314 | 5.76 | 2.67 | 0.25 | 0.1 | 1.5 | 2 | 40 | 40 | 101325 | 2.09 |
| 315 | 5.76 | 2.67 | 0.25 | 0.1 | 1.5 | 2 | 60 | 60 | 101325 | 28.56 |
| 316 | 5.76 | 2.67 | 0.25 | 0.1 | 1.5 | 2 | 80 | 80 | 101325 | 39.02 |
| 317 | 5.76 | 2.67 | 0.25 | 0.1 | 1.5 | 3 | 40 | 40 | 101325 | 3.71 |
| 318 | 5.76 | 2.67 | 0.25 | 0.1 | 1.5 | 3 | 60 | 60 | 101325 | 35.58 |
| 319 | 5.76 | 2.67 | 0.25 | 0.1 | 1.5 | 3 | 80 | 80 | 101325 | 48.87 |
| 320 | 38.22 | 2.32 | 0.28 | 0.077 | 0.34 | 7 | 20 | 20 | 101325 | 14.87 |
| 321 | 38.22 | 2.32 | 0.28 | 0.077 | 0.34 | 28 | 20 | 20 | 101325 | 20.58 |
| 322 | 38.22 | 2.32 | 0.3 | 0.038 | 0 | 7 | 80 | 80 | 101325 | 18.31 |
| 323 | 38.22 | 2.32 | 0.3 | 0.038 | 0 | 28 | 80 | 80 | 101325 | 18.63 |
| 324 | 32 | 2.32 | 0.3 | 0.038 | 0 | 7 | 80 | 80 | 101325 | 14.84 |
| 325 | 32 | 2.32 | 0.3 | 0.038 | 0 | 28 | 80 | 80 | 101325 | 19.57 |
| 326 | 46 | 2.32 | 0.3 | 0.038 | 0 | 7 | 80 | 80 | 101325 | 8.73 |
| 327 | 46 | 2.32 | 0.3 | 0.038 | 0 | 28 | 80 | 80 | 101325 | 13.89 |
| 328 | 38.22 | 2.32 | 0.3 | 0.007 | 0 | 28 | 80 | 80 | 101325 | 0.73 |
| 329 | 38.22 | 2.32 | 0.3 | 0.023 | 0 | 7 | 80 | 80 | 101325 | 3.15 |
| 330 | 38.22 | 2.32 | 0.3 | 0.023 | 0 | 28 | 80 | 80 | 101325 | 7.89 |
| 331 | 38.22 | 2.32 | 0.3 | 0.054 | 0 | 7 | 80 | 80 | 101325 | 44.63 |
| 332 | 38.22 | 2.32 | 0.3 | 0.054 | 0 | 28 | 80 | 80 | 101325 | 48.31 |
| 333 | 103 | 2.2 | 0.19 | 0.095 | 0 | 0.25 | 80 | 80 | 101325 | 16 |
| 334 | 103 | 2.2 | 0.19 | 0.095 | 0 | 4.25 | 80 | -80 | 101325 | 25 |
| 335 | 103 | 2.2 | 0.19 | 0.095 | 0 | 8.25 | 80 | -80 | 101325 | 23.5 |
| 336 | 103 | 2.2 | 0.19 | 0.095 | 0 | 16.25 | 80 | -80 | 101325 | 32.6 |
| 337 | 31.1 | 2.25 | 0.3 | 0.075 | 0 | 1 | 27 | 41 | 101325 | 0.43 |
| 338 | 31.1 | 2.25 | 0.3 | 0.075 | 0.5 | 1 | 27 | 41 | 101325 | 0.3 |
| 339 | 31.1 | 2.25 | 0.3 | 0.075 | 1 | 1 | 27 | 41 | 101325 | 0.4 |
| 340 | 31.1 | 2.25 | 0.3 | 0.075 | 0 | 1 | 55 | 64 | 101325 | 1.54 |
| 341 | 31.1 | 2.25 | 0.3 | 0.075 | 0.5 | 1 | 55 | 64 | 101325 | 0.5 |
| 342 | 31.1 | 2.25 | 0.3 | 0.075 | 1 | 1 | 55 | 64 | 101325 | 0.68 |
| 343 | 31.1 | 2.25 | 0.3 | 0.075 | 0 | 1 | 80 | 84 | 101325 | 12.85 |
| 344 | 31.1 | 2.25 | 0.3 | 0.075 | 0.5 | 1 | 80 | 84 | 101325 | 4.59 |
| 345 | 31.1 | 2.25 | 0.3 | 0.075 | 1 | 1 | 80 | 84 | 101325 | 2.38 |
| 346 | 31.1 | 2.25 | 0.3 | 0.075 | 0 | 1 | 25 | 25 | 10 | 0.49 |
| 347 | 31.1 | 2.25 | 0.3 | 0.075 | 0.5 | 1 | 25 | 25 | 10 | 3.28 |
| 348 | 31.1 | 2.25 | 0.3 | 0.075 | 1 | 1 | 25 | 25 | 10 | 5.03 |
| 349 | 100 | 2.5 | 0.3 | 0.029 | 0 | 1 | 52.7 | 60.2 | 101325 | 0.35 |
| 350 | 100 | 2.5 | 0.3 | 0.059 | 0 | 1 | 52.7 | 60.2 | 101325 | 1.61 |
| 351 | 100 | 2.5 | 0.3 | 0.089 | 0 | 1 | 52.7 | 60.2 | 101325 | 1.62 |
| 352 | 100 | 2.5 | 0.3 | 0.029 | 0 | 3 | 52.7 | 76.2 | 101325 | 2.29 |
| 353 | 100 | 2.5 | 0.3 | 0.059 | 0 | 3 | 52.7 | 76.2 | 101325 | 3.26 |
| 354 | 100 | 2.5 | 0.3 | 0.089 | 0 | 3 | 52.7 | 76.2 | 101325 | 4.34 |
| 355 | 100 | 2.5 | 0.3 | 0.029 | 0 | 1 | 74.9 | 76.2 | 101325 | 1.43 |
| 356 | 100 | 2.5 | 0.3 | 0.059 | 0 | 1 | 74.9 | 76.2 | 101325 | 3.23 |
| 357 | 100 | 2.5 | 0.3 | 0.089 | 0 | 1 | 74.9 | 76.2 | 101325 | 3.13 |
| 358 | 100 | 2.5 | 0.3 | 0.029 | 0 | 3 | 74.9 | 77.5 | 101325 | 1.14 |
| 359 | 100 | 2.5 | 0.3 | 0.089 | 0 | 3 | 74.9 | 77.5 | 101325 | 4.88 |
| 360 | 100 | 2.5 | 0.3 | 0.029 | 0 | 1 | 76.3 | 74.9 | 101325 | 1.44 |
| 361 | 100 | 2.5 | 0.3 | 0.059 | 0 | 1 | 76.3 | 74.9 | 101325 | 3.23 |
| 362 | 100 | 2.5 | 0.3 | 0.089 | 0 | 1 | 76.3 | 74.9 | 101325 | 3.17 |
| 363 | 100 | 2.5 | 0.3 | 0.029 | 0 | 3 | 76.3 | 52.7 | 101325 | 1.94 |
| 364 | 100 | 2.5 | 0.3 | 0.059 | 0 | 3 | 76.3 | 52.7 | 101325 | 5.21 |
| 365 | 100 | 2.5 | 0.3 | 0.089 | 0 | 3 | 76.3 | 52.7 | 101325 | 6.77 |
| 366 | 100 | 2.5 | 0.3 | 0.029 | 0 | 1 | 60.3 | 52.7 | 101325 | 0.68 |
| 367 | 100 | 2.5 | 0.3 | 0.059 | 0 | 1 | 60.3 | 52.7 | 101325 | 1.71 |
| 368 | 100 | 2.5 | 0.3 | 0.089 | 0 | 1 | 60.3 | 52.7 | 101325 | 1.5 |
| 369 | 100 | 2.5 | 0.3 | 0.029 | 0 | 3 | 60.3 | 7.4 | 101325 | 0.79 |
| 370 | 100 | 2.5 | 0.3 | 0.059 | 0 | 3 | 60.3 | 7.4 | 101325 | 2.29 |
| 371 | 100 | 2.5 | 0.3 | 0.089 | 0 | 3 | 60.3 | 7.4 | 101325 | 2.22 |
| 372 | 247.45 | 5.61 | 0.23 | 0.13 | 0.63 | 3 | 76 | 76 | 101325 | 24.59 |
| 373 | 247.45 | 5.61 | 0.23 | 0.13 | 0.63 | 3 | 76 | 76 | 101325 | 31.86 |
| 374 | 138.45 | 2.46 | 0.23 | 0.13 | 0.63 | 3 | 76 | 76 | 101325 | 27.95 |
| 375 | 138.45 | 2.46 | 0.23 | 0.13 | 0.63 | 3 | 76 | 76 | 101325 | 41.23 |

**Supplementary Code**

**Code S1 Stacking-RS model**

import pandas as pd

import numpy as np

import time

import os

import warnings

warnings.filterwarnings('ignore')

from sklearn.model\_selection import train\_test\_split, RandomizedSearchCV

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

from sklearn.linear\_model import LinearRegression

from xgboost import XGBRegressor

from catboost import CatBoostRegressor

from sklearn.ensemble import StackingRegressor

from scipy.stats import randint as sp\_randint

from scipy.stats import uniform

# Basic configuration

os.makedirs('catboost\_info', exist\_ok=True)

TREE\_THREADS = 2 # Control the number of threads for tree-based models to balance speed

# Metric calculation function

def calculate\_metrics(actual, predicted):

r2 = r2\_score(actual, predicted)

rmse = np.sqrt(mean\_squared\_error(actual, predicted))

rrmse = rmse / np.mean(actual) if np.mean(actual) != 0 else np.inf

mae = mean\_absolute\_error(actual, predicted)

rho = rrmse / (1 + np.sqrt(max(r2, 0.001)))

return r2, rmse, rrmse, mae, rho

if \_\_name\_\_ == "\_\_main\_\_":

start\_total = time.perf\_counter()

# 1. Data loading and preprocessing

print("1. Loading data...")

data = pd.read\_excel('CS.xlsx').dropna() # Simple removal of missing values

X = data.iloc[:, 0:9].values

y = data.iloc[:, 9].values

print(f"Dataset size: {X.shape[0]} samples × {X.shape[1]} features\n")

# 2. Split dataset (30% test set)

print("2. Splitting into training set (70%) / test set (30%)...")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=0, shuffle=True

)

# 3. Define six base learners

print("3. Initializing base learners: RF, SVR, MLP, XGB, CatBoost, AdaBoost...\n")

estimators = [

('rf', Pipeline([

('scaler', MinMaxScaler()),

('model', RandomForestRegressor(

random\_state=0, n\_jobs=TREE\_THREADS,

max\_features='sqrt', max\_depth=8

))

])),

('svr', Pipeline([

('scaler', MinMaxScaler()),

('model', SVR(kernel='rbf', cache\_size=100))

])),

('mlp', Pipeline([

('scaler', MinMaxScaler()),

('model', MLPRegressor(

random\_state=0, early\_stopping=True,

max\_iter=300, batch\_size=32, hidden\_layer\_sizes=(30, 20)

))

])),

('xgb', Pipeline([

('scaler', MinMaxScaler()),

('model', XGBRegressor(

random\_state=0, n\_jobs=TREE\_THREADS,

max\_depth=6, subsample=0.8, verbosity=0

))

])),

('cat', Pipeline([

('scaler', MinMaxScaler()),

('model', CatBoostRegressor(

random\_state=0, verbose=0,

train\_dir='catboost\_info', depth=6, thread\_count=TREE\_THREADS

))

])),

('ada', Pipeline([

('scaler', MinMaxScaler()),

('model', AdaBoostRegressor(

random\_state=0,

estimator=DecisionTreeRegressor(max\_depth=3),

n\_estimators=100

))

]))

]

# 4. Build the Stacking model (meta-learner set to LinearRegression)

stacking\_model = StackingRegressor(

estimators=estimators,

# Replaced with LinearRegression

final\_estimator=LinearRegression(),

cv=5, n\_jobs=-1, passthrough=False

)

# 5. Define the parameter search space

param\_dist = {

# RF

'rf\_\_model\_\_n\_estimators': sp\_randint(50, 200),

'rf\_\_model\_\_max\_depth': sp\_randint(4, 8),

# SVR

'svr\_\_model\_\_C': uniform(1, 50),

'svr\_\_model\_\_epsilon': uniform(0.05, 0.3),

# MLP

'mlp\_\_model\_\_hidden\_layer\_sizes': [(30,), (30, 20), (50,)],

'mlp\_\_model\_\_alpha': uniform(0.001, 0.01),

# XGB

'xgb\_\_model\_\_n\_estimators': sp\_randint(50, 200),

'xgb\_\_model\_\_max\_depth': sp\_randint(4, 6),

'xgb\_\_model\_\_learning\_rate': uniform(0.05, 0.15),

# CatBoost

'cat\_\_model\_\_iterations': sp\_randint(50, 200),

'cat\_\_model\_\_depth': sp\_randint(4, 6),

'cat\_\_model\_\_learning\_rate': uniform(0.05, 0.15),

# AdaBoost

'ada\_\_model\_\_n\_estimators': sp\_randint(50, 200),

'ada\_\_model\_\_learning\_rate': uniform(0.05, 0.15),

# LinearRegression

'final\_estimator\_\_fit\_intercept': [True, False]

}

# 6. Randomized search (30 iterations to balance speed and accuracy)

print("4. Starting hyperparameter search (30 iterations + 5-fold cross-validation)...")

random\_search = RandomizedSearchCV(

estimator=stacking\_model,

param\_distributions=param\_dist,

n\_iter=30, cv=5,

scoring='r2', random\_state=0,

verbose=1, n\_jobs=-1

)

# 7. Training (with exception handling)

start\_search = time.perf\_counter()

try:

random\_search.fit(X\_train, y\_train)

best\_model = random\_search.best\_estimator\_

# Key: print the best parameters completely (keep original parameter names for reproducibility)

print("\n Best parameter set (can be used directly for reproducibility):")

for param, value in random\_search.best\_params\_.items():

print(f" {param}: {value}")

except Exception as e:

print(f"\n Training warning: {str(e)}. Retrying with single thread...")

random\_search.n\_jobs = 1

random\_search.fit(X\_train, y\_train)

best\_model = random\_search.best\_estimator\_

end\_search = time.perf\_counter()

# 8. Prediction and saving results

print("\n5. Making predictions...")

start\_predict = time.perf\_counter()

y\_train\_pred = best\_model.predict(X\_train)

y\_test\_pred = best\_model.predict(X\_test)

# Save prediction results for the training set

train\_path = 'Stacking-RS-train.xlsx'

pd.DataFrame({'Actual': y\_train, 'Predicted': y\_train\_pred}).to\_excel(train\_path, index=False)

print(f" Training set predictions saved to: {train\_path}")

# Save prediction results for the test set

test\_path = 'Stacking-RS-test.xlsx'

pd.DataFrame({'Actual': y\_test, 'Predicted': y\_test\_pred}).to\_excel(test\_path, index=False)

print(f" Test set predictions saved to: {test\_path}")

# New: save prediction results for the full dataset

full\_actual = np.concatenate([y\_train, y\_test])

full\_predicted = np.concatenate([y\_train\_pred, y\_test\_pred])

full\_path = 'Stacking-RS-full.xlsx'

pd.DataFrame({'Actual': full\_actual, 'Predicted': full\_predicted}).to\_excel(full\_path, index=False)

print(f" Full-dataset predictions saved to: {full\_path}")

end\_predict = time.perf\_counter()

# 9. Compute and print metrics

metrics\_train = calculate\_metrics(y\_train, y\_train\_pred)

metrics\_test = calculate\_metrics(y\_test, y\_test\_pred)

metrics\_full = calculate\_metrics(

np.concatenate([y\_train, y\_test]),

np.concatenate([y\_train\_pred, y\_test\_pred])

)

# 10. Time and metrics summary

total\_time = time.perf\_counter() - start\_total

search\_time = end\_search - start\_search

predict\_time = end\_predict - start\_predict

print("\n==================== Summary ====================")

print("Time statistics:")

print(f" - Hyperparameter search: {search\_time:.2f} s ({search\_time/60:.2f} min)")

print(f" - Prediction time: {predict\_time:.2f} s")

print(f" - Total runtime: {total\_time:.2f} s ({total\_time/60:.2f} min)\n")

print("Training set metrics:")

print(f" R²={metrics\_train[0]:.4f}, RMSE={metrics\_train[1]:.4f}, RRMSE={metrics\_train[2]:.4f}, MAE={metrics\_train[3]:.4f}, ρ={metrics\_train[4]:.4f}")

print("Test set metrics:")

print(f" R²={metrics\_test[0]:.4f}, RMSE={metrics\_test[1]:.4f}, RRMSE={metrics\_test[2]:.4f}, MAE={metrics\_test[3]:.4f}, ρ={metrics\_test[4]:.4f}")

print("Full dataset metrics:")

print(f" R²={metrics\_full[0]:.4f}, RMSE={metrics\_full[1]:.4f}, RRMSE={metrics\_full[2]:.4f}, MAE={metrics\_full[3]:.4f}, ρ={metrics\_full[4]:.4f}")

print("=================================================")

**Code S2 Stacking-BO model**

import pandas as pd

import numpy as np

import time

import os

import warnings

warnings.filterwarnings('ignore')

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor, StackingRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LinearRegression

from xgboost import XGBRegressor

from catboost import CatBoostRegressor

from bayes\_opt import BayesianOptimization

# Basic Configuration

os.makedirs('catboost\_info', exist\_ok=True)

TREE\_THREADS = 2 # Control tree-based model threads for speed balance

# Construct Stacking Ensemble Model

def build\_stacking\_model(

rf\_n\_estimators, rf\_max\_depth,

svr\_C, svr\_epsilon,

mlp\_hidden\_idx, mlp\_alpha,

xgb\_n\_estimators, xgb\_max\_depth, xgb\_learning\_rate,

cat\_iterations, cat\_depth, cat\_learning\_rate,

ada\_n\_estimators, ada\_learning\_rate,

final\_fit\_intercept

):

# Mapping table for MLP hidden layer structure

hidden\_map = {

0: (30,), # Index 0 -> Single hidden layer with 30 neurons

1: (30, 20), # Index 1 -> Double hidden layers with 30→20 neurons

2: (50,) # Index 2 -> Single hidden layer with 50 neurons

}

mlp\_hidden = hidden\_map[int(mlp\_hidden\_idx)]

# Base estimators with independent data scaling pipeline

base\_estimators = [

('rf', Pipeline([

('scaler', MinMaxScaler()),

('model', RandomForestRegressor(

random\_state=0, n\_jobs=TREE\_THREADS, max\_features='sqrt',

n\_estimators=int(rf\_n\_estimators), max\_depth=int(rf\_max\_depth)

))

])),

('svr', Pipeline([

('scaler', MinMaxScaler()),

('model', SVR(kernel='rbf', cache\_size=100, C=svr\_C, epsilon=svr\_epsilon))

])),

('mlp', Pipeline([

('scaler', MinMaxScaler()),

('model', MLPRegressor(

random\_state=0, early\_stopping=True, max\_iter=300, batch\_size=32,

hidden\_layer\_sizes=mlp\_hidden, alpha=mlp\_alpha

))

])),

('xgb', Pipeline([

('scaler', MinMaxScaler()),

('model', XGBRegressor(

random\_state=0, n\_jobs=TREE\_THREADS, subsample=0.8, verbosity=0,

n\_estimators=int(xgb\_n\_estimators), max\_depth=int(xgb\_max\_depth), learning\_rate=xgb\_learning\_rate

))

])),

('cat', Pipeline([

('scaler', MinMaxScaler()),

('model', CatBoostRegressor(

random\_state=0, verbose=0, train\_dir='catboost\_info', thread\_count=TREE\_THREADS,

iterations=int(cat\_iterations), depth=int(cat\_depth), learning\_rate=cat\_learning\_rate

))

])),

('ada', Pipeline([

('scaler', MinMaxScaler()),

('model', AdaBoostRegressor(

random\_state=0, estimator=DecisionTreeRegressor(max\_depth=3),

n\_estimators=int(ada\_n\_estimators), learning\_rate=ada\_learning\_rate

))

]))

]

# Return assembled Stacking Regressor

return StackingRegressor(

estimators=base\_estimators,

final\_estimator=LinearRegression(fit\_intercept=final\_fit\_intercept),

cv=3, n\_jobs=-1, passthrough=False

)

# Bayesian Optimization Objective Function (Core Module)

def bayesian\_optimization\_objective(

rf\_n\_estimators, rf\_max\_depth,

svr\_C, svr\_epsilon,

mlp\_hidden\_idx, mlp\_alpha,

xgb\_n\_estimators, xgb\_max\_depth, xgb\_learning\_rate,

cat\_iterations, cat\_depth, cat\_learning\_rate,

ada\_n\_estimators, ada\_learning\_rate,

final\_fit\_intercept

):

try:

# Initialize stacking model with hyperparameters from Bayesian search

current\_model = build\_stacking\_model(

rf\_n\_estimators, rf\_max\_depth,

svr\_C, svr\_epsilon,

mlp\_hidden\_idx, mlp\_alpha,

xgb\_n\_estimators, xgb\_max\_depth, xgb\_learning\_rate,

cat\_iterations, cat\_depth, cat\_learning\_rate,

ada\_n\_estimators, ada\_learning\_rate,

final\_fit\_intercept

)

cv\_scores = cross\_val\_score(current\_model, X\_train, y\_train, cv=5, scoring='r2', n\_jobs=-1)

return cv\_scores.mean()

except Exception:

return -1 # Return penalty value for failed model training

if \_\_name\_\_ == "\_\_main\_\_":

start\_time = time.perf\_counter()

# Data Loading & Dataset Split (Minimal reserved necessary code)

dataset = pd.read\_excel('CS.xlsx').dropna()

X = dataset.iloc[:, 0:9].values

y = dataset.iloc[:, 9].values

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=0, shuffle=True

)

print(f"Dataset Scale: {X.shape[0]} Samples × {X.shape[1]} Features")

# Define Bayesian Optimization Hyperparameter Search Space (Core)

hyperparameter\_search\_bounds = {

# Random Forest Hyperparameters

'rf\_n\_estimators': (50, 200),

'rf\_max\_depth': (4, 8),

# SVR Hyperparameters

'svr\_C': (1, 50),

'svr\_epsilon': (0.05, 0.3),

# MLP Hyperparameters

'mlp\_hidden\_idx': (0, 2),

'mlp\_alpha': (0.001, 0.01),

# XGBoost Hyperparameters

'xgb\_n\_estimators': (50, 200),

'xgb\_max\_depth': (4, 6),

'xgb\_learning\_rate': (0.05, 0.15),

# CatBoost Hyperparameters

'cat\_iterations': (50, 200),

'cat\_depth': (4, 6),

'cat\_learning\_rate': (0.05, 0.15),

# AdaBoost Hyperparameters

'ada\_n\_estimators': (50, 200),

'ada\_learning\_rate': (0.05, 0.15),

# Stacking Meta-model Hyperparameter

'final\_fit\_intercept': (0, 1)

}

# Initialize Bayesian Optimizer

bo\_optimizer = BayesianOptimization(

f=bayesian\_optimization\_objective,

pbounds=hyperparameter\_search\_bounds,

random\_state=0,

verbose=2

)

# Execute Bayesian Optimization (5 random init points + 30 iterative optimizations)

bo\_optimizer.maximize(init\_points=5, n\_iter=30)

# Extract & Format Optimal Hyperparameters

optimal\_hyperparams = bo\_optimizer.max['params']

print("\n Optimal Hyperparameter Combination (Raw Values):")

for param\_name, param\_value in optimal\_hyperparams.items():

print(f" {param\_name}: {param\_value}")

# Type Conversion for Integer/Bool Constrained Hyperparameters

optimal\_hyperparams['rf\_n\_estimators'] = int(optimal\_hyperparams['rf\_n\_estimators'])

optimal\_hyperparams['rf\_max\_depth'] = int(optimal\_hyperparams['rf\_max\_depth'])

optimal\_hyperparams['mlp\_hidden\_idx'] = int(optimal\_hyperparams['mlp\_hidden\_idx'])

optimal\_hyperparams['xgb\_n\_estimators'] = int(optimal\_hyperparams['xgb\_n\_estimators'])

optimal\_hyperparams['xgb\_max\_depth'] = int(optimal\_hyperparams['xgb\_max\_depth'])

optimal\_hyperparams['cat\_iterations'] = int(optimal\_hyperparams['cat\_iterations'])

optimal\_hyperparams['cat\_depth'] = int(optimal\_hyperparams['cat\_depth'])

optimal\_hyperparams['ada\_n\_estimators'] = int(optimal\_hyperparams['ada\_n\_estimators'])

optimal\_hyperparams['final\_fit\_intercept'] = bool(round(optimal\_hyperparams['final\_fit\_intercept']))

# Train the Final Optimal Stacking Model

optimal\_stacking\_model = build\_stacking\_model(\*\*optimal\_hyperparams)

optimal\_stacking\_model.fit(X\_train, y\_train)

# Model Prediction

y\_train\_pred = optimal\_stacking\_model.predict(X\_train)

y\_test\_pred = optimal\_stacking\_model.predict(X\_test)

total\_running\_time = time.perf\_counter() - start\_time

print(f"\nTotal Running Time: {total\_running\_time:.2f}s ({total\_running\_time/60:.2f} mins)")

**Code S3 Stacking-PSO model**

import pandas as pd

import numpy as np

import time

import os

import warnings

import random

warnings.filterwarnings('ignore')

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

from sklearn.linear\_model import LinearRegression

from xgboost import XGBRegressor

from catboost import CatBoostRegressor

from sklearn.ensemble import StackingRegressor

from pyswarm import pso # Particle Swarm Optimization library

# Basic configuration

os.makedirs('catboost\_info', exist\_ok=True)

TREE\_THREADS = 2

# Fix random seeds manually (PSO does not support random\_state)

random.seed(0)

np.random.seed(0)

# Metric calculation function

def calculate\_metrics(actual, predicted):

r2 = r2\_score(actual, predicted)

rmse = np.sqrt(mean\_squared\_error(actual, predicted))

rrmse = rmse / np.mean(actual) if np.mean(actual) != 0 else np.inf

mae = mean\_absolute\_error(actual, predicted)

rho = rrmse / (1 + np.sqrt(max(r2, 0.001)))

return r2, rmse, rrmse, mae, rho

# Categorical parameter mappings

hlayer\_map = {0: (30,), 1: (30, 20), 2: (50,)}

fit\_intercept\_map = {0: False, 1: True}

# PSO parameter bounds (lb: lower bound, ub: upper bound)

lb = [

50, 4, # RF: n\_estimators, max\_depth

1, 0.05, # SVR: C, epsilon

0, 0.001, # MLP: hidden\_idx, alpha

50, 4, 0.05, # XGB: n\_estimators, max\_depth, learning\_rate

50, 4, 0.05, # CatBoost: iterations, depth, learning\_rate

50, 0.05, # AdaBoost: n\_estimators, learning\_rate

0 # Meta-learner: fit\_intercept index

]

ub = [

200, 8,

50, 0.3,

2, 0.01,

200, 6, 0.15,

200, 6, 0.15,

200, 0.15,

1

]

# PSO objective function

def pso\_objective(x):

try:

params = {

'rf\_n\_estimators': int(round(x[0])),

'rf\_max\_depth': int(round(x[1])),

'svr\_C': x[2],

'svr\_epsilon': x[3],

'mlp\_hidden': hlayer\_map[int(round(x[4]))],

'mlp\_alpha': x[5],

'xgb\_n\_estimators': int(round(x[6])),

'xgb\_max\_depth': int(round(x[7])),

'xgb\_learning\_rate': x[8],

'cat\_iterations': int(round(x[9])),

'cat\_depth': int(round(x[10])),

'cat\_learning\_rate': x[11],

'ada\_n\_estimators': int(round(x[12])),

'ada\_learning\_rate': x[13],

'final\_fit\_intercept': fit\_intercept\_map[int(round(x[14]))]

}

estimators = [

('rf', Pipeline([

('scaler', MinMaxScaler()),

('model', RandomForestRegressor(

random\_state=0, n\_jobs=TREE\_THREADS,

max\_features='sqrt',

n\_estimators=params['rf\_n\_estimators'],

max\_depth=params['rf\_max\_depth']

))

])),

('svr', Pipeline([

('scaler', MinMaxScaler()),

('model', SVR(

kernel='rbf', cache\_size=100,

C=params['svr\_C'], epsilon=params['svr\_epsilon']

))

])),

('mlp', Pipeline([

('scaler', MinMaxScaler()),

('model', MLPRegressor(

random\_state=0, early\_stopping=True,

max\_iter=300, batch\_size=32,

hidden\_layer\_sizes=params['mlp\_hidden'],

alpha=params['mlp\_alpha']

))

])),

('xgb', Pipeline([

('scaler', MinMaxScaler()),

('model', XGBRegressor(

random\_state=0, n\_jobs=TREE\_THREADS,

subsample=0.8, verbosity=0,

n\_estimators=params['xgb\_n\_estimators'],

max\_depth=params['xgb\_max\_depth'],

learning\_rate=params['xgb\_learning\_rate']

))

])),

('cat', Pipeline([

('scaler', MinMaxScaler()),

('model', CatBoostRegressor(

random\_state=0, verbose=0,

train\_dir='catboost\_info',

thread\_count=TREE\_THREADS,

iterations=params['cat\_iterations'],

depth=params['cat\_depth'],

learning\_rate=params['cat\_learning\_rate']

))

])),

('ada', Pipeline([

('scaler', MinMaxScaler()),

('model', AdaBoostRegressor(

random\_state=0,

estimator=DecisionTreeRegressor(max\_depth=3),

n\_estimators=params['ada\_n\_estimators'],

learning\_rate=params['ada\_learning\_rate']

))

]))

]

stacking\_model = StackingRegressor(

estimators=estimators,

final\_estimator=LinearRegression(

fit\_intercept=params['final\_fit\_intercept']

),

cv=5, n\_jobs=-1, passthrough=False

)

cv\_scores = cross\_val\_score(

stacking\_model, X\_train, y\_train,

cv=5, scoring='r2', n\_jobs=-1

)

return -cv\_scores.mean()

except Exception:

return 1000.0

if \_\_name\_\_ == "\_\_main\_\_":

start\_total = time.perf\_counter()

# 1. Data loading and preprocessing

print("1. Loading data...")

data = pd.read\_excel('CS.xlsx').dropna()

X = data.iloc[:, 0:9].values

y = data.iloc[:, 9].values

print(f"Dataset size: {X.shape[0]} samples × {X.shape[1]} features\n")

# 2. Train–test split

print("2. Splitting into training set (70%) and test set (30%)...")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.3, random\_state=0, shuffle=True

)

# 3. Particle Swarm Optimization

print("3. Running Particle Swarm Optimization (PSO)...")

start\_search = time.perf\_counter()

x\_opt, f\_opt = pso(

func=pso\_objective,

lb=lb,

ub=ub,

swarmsize=20,

maxiter=50,

debug=True

)

end\_search = time.perf\_counter()

search\_time = end\_search - start\_search

# 4. Extract best parameters

best\_params = {

'rf\_\_model\_\_n\_estimators': int(round(x\_opt[0])),

'rf\_\_model\_\_max\_depth': int(round(x\_opt[1])),

'svr\_\_model\_\_C': x\_opt[2],

'svr\_\_model\_\_epsilon': x\_opt[3],

'mlp\_\_model\_\_hidden\_layer\_sizes': hlayer\_map[int(round(x\_opt[4]))],

'mlp\_\_model\_\_alpha': x\_opt[5],

'xgb\_\_model\_\_n\_estimators': int(round(x\_opt[6])),

'xgb\_\_model\_\_max\_depth': int(round(x\_opt[7])),

'xgb\_\_model\_\_learning\_rate': x\_opt[8],

'cat\_\_model\_\_iterations': int(round(x\_opt[9])),

'cat\_\_model\_\_depth': int(round(x\_opt[10])),

'cat\_\_model\_\_learning\_rate': x\_opt[11],

'ada\_\_model\_\_n\_estimators': int(round(x\_opt[12])),

'ada\_\_model\_\_learning\_rate': x\_opt[13],

'final\_estimator\_\_fit\_intercept': fit\_intercept\_map[int(round(x\_opt[14]))]

}

print("\nBest hyperparameter configuration:")

for param, value in best\_params.items():

print(f" {param}: {value}")

# 5. Build and train the final model

estimators = [

('rf', Pipeline([('scaler', MinMaxScaler()), ('model', RandomForestRegressor(random\_state=0, n\_jobs=TREE\_THREADS, max\_features='sqrt'))])),

('svr', Pipeline([('scaler', MinMaxScaler()), ('model', SVR(kernel='rbf', cache\_size=100))])),

('mlp', Pipeline([('scaler', MinMaxScaler()), ('model', MLPRegressor(random\_state=0, early\_stopping=True, max\_iter=300, batch\_size=32))])),

('xgb', Pipeline([('scaler', MinMaxScaler()), ('model', XGBRegressor(random\_state=0, n\_jobs=TREE\_THREADS, subsample=0.8, verbosity=0))])),

('cat', Pipeline([('scaler', MinMaxScaler()), ('model', CatBoostRegressor(random\_state=0, verbose=0, train\_dir='catboost\_info', thread\_count=TREE\_THREADS))])),

('ada', Pipeline([('scaler', MinMaxScaler()), ('model', AdaBoostRegressor(random\_state=0, estimator=DecisionTreeRegressor(max\_depth=3)))]))

]

best\_model = StackingRegressor(

estimators=estimators,

final\_estimator=LinearRegression(),

cv=5, n\_jobs=-1, passthrough=False

).set\_params(\*\*best\_params)

best\_model.fit(X\_train, y\_train)

# 6. Prediction and result saving

print("\n4. Making predictions...")

start\_predict = time.perf\_counter()

y\_train\_pred = best\_model.predict(X\_train)

y\_test\_pred = best\_model.predict(X\_test)

pd.DataFrame({'Actual': y\_train, 'Predicted': y\_train\_pred}).to\_excel('Stacking-PSO-train.xlsx', index=False)

pd.DataFrame({'Actual': y\_test, 'Predicted': y\_test\_pred}).to\_excel('Stacking-PSO-test.xlsx', index=False)

full\_actual = np.concatenate([y\_train, y\_test])

full\_predicted = np.concatenate([y\_train\_pred, y\_test\_pred])

pd.DataFrame({'Actual': full\_actual, 'Predicted': full\_predicted}).to\_excel('Stacking-PSO-full.xlsx', index=False)

end\_predict = time.perf\_counter()

predict\_time = end\_predict - start\_predict

# 7. Metrics and time summary

metrics\_train = calculate\_metrics(y\_train, y\_train\_pred)

metrics\_test = calculate\_metrics(y\_test, y\_test\_pred)

metrics\_full = calculate\_metrics(full\_actual, full\_predicted)

total\_time = time.perf\_counter() - start\_total

print("\n==================== Summary ====================")

print("Time statistics:")

print(f" - Particle Swarm Optimization: {search\_time:.2f} s ({search\_time/60:.2f} min)")

print(f" - Prediction time: {predict\_time:.2f} s")

print(f" - Total runtime: {total\_time:.2f} s ({total\_time/60:.2f} min)\n")

print("Training set metrics:")

print(f" R²={metrics\_train[0]:.4f}, RMSE={metrics\_train[1]:.4f}, RRMSE={metrics\_train[2]:.4f}, MAE={metrics\_train[3]:.4f}, ρ={metrics\_train[4]:.4f}")

print("Test set metrics:")

print(f" R²={metrics\_test[0]:.4f}, RMSE={metrics\_test[1]:.4f}, RRMSE={metrics\_test[2]:.4f}, MAE={metrics\_test[3]:.4f}, ρ={metrics\_test[4]:.4f}")

print("Full dataset metrics:")

print(f" R²={metrics\_full[0]:.4f}, RMSE={metrics\_full[1]:.4f}, RRMSE={metrics\_full[2]:.4f}, MAE={metrics\_full[3]:.4f}, ρ={metrics\_full[4]:.4f}")

print("=================================================")

**Code S4 Stacking-SA model**

import pandas as pd

import numpy as np

import time

import os

import warnings

import random

warnings.filterwarnings('ignore')

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

from sklearn.linear\_model import LinearRegression

from xgboost import XGBRegressor

from sklearn.ensemble import StackingRegressor

from catboost import CatBoostRegressor

os.makedirs('catboost\_info', exist\_ok=True)

TREE\_THREADS = 2

random.seed(0)

np.random.seed(0)

def calculate\_metrics(actual, predicted):

r2 = r2\_score(actual, predicted)

rmse = np.sqrt(mean\_squared\_error(actual, predicted))

rrmse = rmse / np.mean(actual) if np.mean(actual) != 0 else np.inf

mae = mean\_absolute\_error(actual, predicted)

rho = rrmse / (1 + np.sqrt(max(r2, 0.001)))

return r2, rmse, rrmse, mae, rho

hlayer\_map = {0: (30,), 1: (30, 20), 2: (50,)}

fit\_intercept\_map = {0: False, 1: True}

lb = [

50, 4,

1, 0.05,

0, 0.001,

50, 4, 0.05,

50, 4, 0.05,

50, 0.05,

0

]

ub = [

200, 8,

50, 0.3,

2, 0.01,

200, 6, 0.15,

200, 6, 0.15,

200, 0.15,

1

]

def sa\_objective(x):

try:

params = {

'rf\_n\_estimators': int(round(x[0])),

'rf\_max\_depth': int(round(x[1])),

'svr\_C': x[2],

'svr\_epsilon': x[3],

'mlp\_hidden': hlayer\_map[int(round(x[4]))],

'mlp\_alpha': x[5],

'xgb\_n\_estimators': int(round(x[6])),

'xgb\_max\_depth': int(round(x[7])),

'xgb\_learning\_rate': x[8],

'cat\_iterations': int(round(x[9])),

'cat\_depth': int(round(x[10])),

'cat\_learning\_rate': x[11],

'ada\_n\_estimators': int(round(x[12])),

'ada\_learning\_rate': x[13],

'final\_fit\_intercept': fit\_intercept\_map[int(round(x[14]))]

}

estimators = [

('rf', Pipeline([

('scaler', MinMaxScaler()),

('model', RandomForestRegressor(

random\_state=0,

n\_jobs=TREE\_THREADS,

max\_features='sqrt',

n\_estimators=params['rf\_n\_estimators'],

max\_depth=params['rf\_max\_depth']

))

])),

('svr', Pipeline([

('scaler', MinMaxScaler()),

('model', SVR(

kernel='rbf',

cache\_size=100,

C=params['svr\_C'],

epsilon=params['svr\_epsilon']

))

])),

('mlp', Pipeline([

('scaler', MinMaxScaler()),

('model', MLPRegressor(

random\_state=0,

early\_stopping=True,

max\_iter=300,

batch\_size=32,

hidden\_layer\_sizes=params['mlp\_hidden'],

alpha=params['mlp\_alpha']

))

])),

('xgb', Pipeline([

('scaler', MinMaxScaler()),

('model', XGBRegressor(

random\_state=0,

n\_jobs=TREE\_THREADS,

subsample=0.8,

verbosity=0,

n\_estimators=params['xgb\_n\_estimators'],

max\_depth=params['xgb\_max\_depth'],

learning\_rate=params['xgb\_learning\_rate']

))

])),

('cat', Pipeline([

('scaler', MinMaxScaler()),

('model', CatBoostRegressor(

random\_state=0,

verbose=0,

train\_dir='catboost\_info',

thread\_count=TREE\_THREADS,

iterations=params['cat\_iterations'],

depth=params['cat\_depth'],

learning\_rate=params['cat\_learning\_rate']

))

])),

('ada', Pipeline([

('scaler', MinMaxScaler()),

('model', AdaBoostRegressor(

random\_state=0,

estimator=DecisionTreeRegressor(max\_depth=3),

n\_estimators=params['ada\_n\_estimators'],

learning\_rate=params['ada\_learning\_rate']

))

]))

]

stacking\_model = StackingRegressor(

estimators=estimators,

final\_estimator=LinearRegression(fit\_intercept=params['final\_fit\_intercept']),

cv=5,

n\_jobs=-1,

passthrough=False

)

cv\_scores = cross\_val\_score(

stacking\_model,

X\_train, y\_train,

cv=5,

scoring='r2',

n\_jobs=-1

)

return -cv\_scores.mean()

except Exception as e:

return 1000.0

def simulated\_annealing(func, lb, ub, dim, max\_iter=100, init\_temp=100, cool\_rate=0.95, debug=True):

current\_x = np.zeros(dim)

for i in range(dim):

current\_x[i] = np.random.uniform(low=lb[i], high=ub[i])

current\_val = func(current\_x)

best\_x = current\_x.copy()

best\_val = current\_val.copy()

temp = init\_temp

for iter in range(max\_iter):

new\_x = current\_x + temp \* np.random.normal(loc=0, scale=1, size=dim)

new\_x = np.clip(new\_x, lb, ub)

new\_val = func(new\_x)

delta\_E = new\_val - current\_val

if delta\_E < 0:

current\_x = new\_x.copy()

current\_val = new\_val.copy()

if current\_val < best\_val:

best\_val = current\_val

best\_x = current\_x.copy()

else:

accept\_prob = np.exp(-delta\_E / temp)

if np.random.rand() < accept\_prob:

current\_x = new\_x.copy()

current\_val = new\_val.copy()

temp \*= cool\_rate

if debug:

print(f"SA iteration {iter+1}/{max\_iter} | Temperature: {temp:.4f} | Current objective: {current\_val:.6f} | Best objective: {best\_val:.6f}")

return best\_x, best\_val

if \_\_name\_\_ == "\_\_main\_\_":

start\_total = time.perf\_counter()

print("1. Loading data...")

data = pd.read\_excel('CS.xlsx').dropna()

X = data.iloc[:, 0:9].values

y = data.iloc[:, 9].values

print(f"Dataset size: {X.shape[0]} samples × {X.shape[1]} features\n")

print("2. Splitting training set (70%) and test set (30%)...")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y,

test\_size=0.3,

random\_state=0,

shuffle=True

)

print("3. Running simulated annealing optimization (SA)...")

start\_search = time.perf\_counter()

best\_params\_vec, best\_score = simulated\_annealing(

func=sa\_objective,

lb=lb,

ub=ub,

dim=len(lb),

max\_iter=100,

init\_temp=100,

cool\_rate=0.95,

debug=True

)

end\_search = time.perf\_counter()

search\_time = end\_search - start\_search

best\_params = {

'rf\_\_model\_\_n\_estimators': int(round(best\_params\_vec[0])),

'rf\_\_model\_\_max\_depth': int(round(best\_params\_vec[1])),

'svr\_\_model\_\_C': best\_params\_vec[2],

'svr\_\_model\_\_epsilon': best\_params\_vec[3],

'mlp\_\_model\_\_hidden\_layer\_sizes': hlayer\_map[int(round(best\_params\_vec[4]))],

'mlp\_\_model\_\_alpha': best\_params\_vec[5],

'xgb\_\_model\_\_n\_estimators': int(round(best\_params\_vec[6])),

'xgb\_\_model\_\_max\_depth': int(round(best\_params\_vec[7])),

'xgb\_\_model\_\_learning\_rate': best\_params\_vec[8],

'cat\_\_model\_\_iterations': int(round(best\_params\_vec[9])),

'cat\_\_model\_\_depth': int(round(best\_params\_vec[10])),

'cat\_\_model\_\_learning\_rate': best\_params\_vec[11],

'ada\_\_model\_\_n\_estimators': int(round(best\_params\_vec[12])),

'ada\_\_model\_\_learning\_rate': best\_params\_vec[13],

'final\_estimator\_\_fit\_intercept': fit\_intercept\_map[int(round(best\_params\_vec[14]))]

}

print("\nBest hyperparameter configuration:")

for param, value in best\_params.items():

print(f" {param}: {value}")

estimators = [

('rf', Pipeline([('scaler', MinMaxScaler()), ('model', RandomForestRegressor(random\_state=0, n\_jobs=TREE\_THREADS, max\_features='sqrt'))])),

('svr', Pipeline([('scaler', MinMaxScaler()), ('model', SVR(kernel='rbf', cache\_size=100))])),

('mlp', Pipeline([('scaler', MinMaxScaler()), ('model', MLPRegressor(random\_state=0, early\_stopping=True, max\_iter=300, batch\_size=32))])),

('xgb', Pipeline([('scaler', MinMaxScaler()), ('model', XGBRegressor(random\_state=0, n\_jobs=TREE\_THREADS, subsample=0.8, verbosity=0))])),

('cat', Pipeline([('scaler', MinMaxScaler()), ('model', CatBoostRegressor(random\_state=0, verbose=0, train\_dir='catboost\_info', thread\_count=TREE\_THREADS))])),

('ada', Pipeline([('scaler', MinMaxScaler()), ('model', AdaBoostRegressor(random\_state=0, estimator=DecisionTreeRegressor(max\_depth=3)))]))

]

best\_model = StackingRegressor(

estimators=estimators,

final\_estimator=LinearRegression(),

cv=5, n\_jobs=-1, passthrough=False

).set\_params(\*\* best\_params)

best\_model.fit(X\_train, y\_train)

print("\n4. Predicting...")

start\_predict = time.perf\_counter()

y\_train\_pred = best\_model.predict(X\_train)

y\_test\_pred = best\_model.predict(X\_test)

end\_predict = time.perf\_counter()

predict\_time = end\_predict - start\_predict

train\_path = 'Stacking-SA-train.xlsx'

pd.DataFrame({'Actual': y\_train, 'Predicted': y\_train\_pred}).to\_excel(train\_path, index=False)

print(f" Training predictions saved: {train\_path}")

test\_path = 'Stacking-SA-test.xlsx'

pd.DataFrame({'Actual': y\_test, 'Predicted': y\_test\_pred}).to\_excel(test\_path, index=False)

print(f" Test predictions saved: {test\_path}")

full\_actual = np.concatenate([y\_train, y\_test])

full\_predicted = np.concatenate([y\_train\_pred, y\_test\_pred])

full\_path = 'Stacking-SA-full.xlsx'

pd.DataFrame({'Actual': full\_actual, 'Predicted': full\_predicted}).to\_excel(full\_path, index=False)

print(f" Full predictions saved: {full\_path}")

metrics\_train = calculate\_metrics(y\_train, y\_train\_pred)

metrics\_test = calculate\_metrics(y\_test, y\_test\_pred)

metrics\_full = calculate\_metrics(full\_actual, full\_predicted)

total\_time = time.perf\_counter() - start\_total

print("\n==================== Summary ====================")

print("Time statistics:")

print(f" - Simulated annealing (SA): {search\_time:.2f} s ({search\_time/60:.2f} min)")

print(f" - Prediction time: {predict\_time:.2f} s")

print(f" - Total runtime: {total\_time:.2f} s ({total\_time/60:.2f} min)\n")

print("Training set metrics:")

print(f" R²={metrics\_train[0]:.4f}, RMSE={metrics\_train[1]:.4f}, RRMSE={metrics\_train[2]:.4f}, MAE={metrics\_train[3]:.4f}, ρ={metrics\_train[4]:.4f}")

print("Test set metrics:")

print(f" R²={metrics\_test[0]:.4f}, RMSE={metrics\_test[1]:.4f}, RRMSE={metrics\_test[2]:.4f}, MAE={metrics\_test[3]:.4f}, ρ={metrics\_test[4]:.4f}")

print("Full dataset metrics:")

print(f" R²={metrics\_full[0]:.4f}, RMSE={metrics\_full[1]:.4f}, RRMSE={metrics\_full[2]:.4f}, MAE={metrics\_full[3]:.4f}, ρ={metrics\_full[4]:.4f}")

print("=================================================")

**Code S5 Stacking-GWO model**

import pandas as pd

import numpy as np

import time

import os

import warnings

import random

warnings.filterwarnings('ignore')

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

from sklearn.linear\_model import LinearRegression

from xgboost import XGBRegressor

from catboost import CatBoostRegressor

from sklearn.ensemble import StackingRegressor

os.makedirs('catboost\_info', exist\_ok=True)

TREE\_THREADS = 2

random.seed(0)

np.random.seed(0)

def calculate\_metrics(actual, predicted):

r2 = r2\_score(actual, predicted)

rmse = np.sqrt(mean\_squared\_error(actual, predicted))

rrmse = rmse / np.mean(actual) if np.mean(actual) != 0 else np.inf

mae = mean\_absolute\_error(actual, predicted)

rho = rrmse / (1 + np.sqrt(max(r2, 0.001)))

return r2, rmse, rrmse, mae, rho

hlayer\_map = {0: (30,), 1: (30, 20), 2: (50,)}

fit\_intercept\_map = {0: False, 1: True}

lb = [

50, 4,

1, 0.05,

0, 0.001,

50, 4, 0.05,

50, 4, 0.05,

50, 0.05,

0

]

ub = [

200, 8,

50, 0.3,

2, 0.01,

200, 6, 0.15,

200, 6, 0.15,

200, 0.15,

1

]

def gwo\_objective(x):

try:

params = {

'rf\_n\_estimators': int(round(x[0])),

'rf\_max\_depth': int(round(x[1])),

'svr\_C': x[2],

'svr\_epsilon': x[3],

'mlp\_hidden': hlayer\_map[int(round(x[4]))],

'mlp\_alpha': x[5],

'xgb\_n\_estimators': int(round(x[6])),

'xgb\_max\_depth': int(round(x[7])),

'xgb\_learning\_rate': x[8],

'cat\_iterations': int(round(x[9])),

'cat\_depth': int(round(x[10])),

'cat\_learning\_rate': x[11],

'ada\_n\_estimators': int(round(x[12])),

'ada\_learning\_rate': x[13],

'final\_fit\_intercept': fit\_intercept\_map[int(round(x[14]))]

}

estimators = [

('rf', Pipeline([

('scaler', MinMaxScaler()),

('model', RandomForestRegressor(

random\_state=0,

n\_jobs=TREE\_THREADS,

max\_features='sqrt',

n\_estimators=params['rf\_n\_estimators'],

max\_depth=params['rf\_max\_depth']

))

])),

('svr', Pipeline([

('scaler', MinMaxScaler()),

('model', SVR(

kernel='rbf',

cache\_size=100,

C=params['svr\_C'],

epsilon=params['svr\_epsilon']

))

])),

('mlp', Pipeline([

('scaler', MinMaxScaler()),

('model', MLPRegressor(

random\_state=0,

early\_stopping=True,

max\_iter=300,

batch\_size=32,

hidden\_layer\_sizes=params['mlp\_hidden'],

alpha=params['mlp\_alpha']

))

])),

('xgb', Pipeline([

('scaler', MinMaxScaler()),

('model', XGBRegressor(

random\_state=0,

n\_jobs=TREE\_THREADS,

subsample=0.8,

verbosity=0,

n\_estimators=params['xgb\_n\_estimators'],

max\_depth=params['xgb\_max\_depth'],

learning\_rate=params['xgb\_learning\_rate']

))

])),

('cat', Pipeline([

('scaler', MinMaxScaler()),

('model', CatBoostRegressor(

random\_state=0,

verbose=0,

train\_dir='catboost\_info',

thread\_count=TREE\_THREADS,

iterations=params['cat\_iterations'],

depth=params['cat\_depth'],

learning\_rate=params['cat\_learning\_rate']

))

])),

('ada', Pipeline([

('scaler', MinMaxScaler()),

('model', AdaBoostRegressor(

random\_state=0,

estimator=DecisionTreeRegressor(max\_depth=3),

n\_estimators=params['ada\_n\_estimators'],

learning\_rate=params['ada\_learning\_rate']

))

]))

]

stacking\_model = StackingRegressor(

estimators=estimators,

final\_estimator=LinearRegression(fit\_intercept=params['final\_fit\_intercept']),

cv=5,

n\_jobs=-1,

passthrough=False

)

cv\_scores = cross\_val\_score(

stacking\_model, X\_train, y\_train,

cv=5, scoring='r2', n\_jobs=-1

)

return -cv\_scores.mean()

except Exception as e:

return 1000.0

def gwo(func, lb, ub, dim, search\_agents=20, max\_iter=50, debug=True):

positions = np.zeros((search\_agents, dim))

for i in range(dim):

positions[:, i] = np.random.uniform(low=lb[i], high=ub[i], size=search\_agents)

alpha\_pos = np.zeros(dim)

alpha\_score = float('inf')

beta\_pos = np.zeros(dim)

beta\_score = float('inf')

delta\_pos = np.zeros(dim)

delta\_score = float('inf')

for iter in range(max\_iter):

for i in range(search\_agents):

for j in range(dim):

positions[i, j] = np.clip(positions[i, j], lb[j], ub[j])

score = func(positions[i, :])

if score < alpha\_score:

alpha\_score = score

alpha\_pos = positions[i, :].copy()

elif score < beta\_score and score > alpha\_score:

beta\_score = score

beta\_pos = positions[i, :].copy()

elif score < delta\_score and score > beta\_score:

delta\_score = score

delta\_pos = positions[i, :].copy()

a = 2 - iter \* (2 / max\_iter)

for i in range(search\_agents):

for j in range(dim):

r1 = np.random.rand()

r2 = np.random.rand()

A1 = 2 \* a \* r1 - a

C1 = 2 \* r2

D\_alpha = abs(C1 \* alpha\_pos[j] - positions[i, j])

X1 = alpha\_pos[j] - A1 \* D\_alpha

r1 = np.random.rand()

r2 = np.random.rand()

A2 = 2 \* a \* r1 - a

C2 = 2 \* r2

D\_beta = abs(C2 \* beta\_pos[j] - positions[i, j])

X2 = beta\_pos[j] - A2 \* D\_beta

r1 = np.random.rand()

r2 = np.random.rand()

A3 = 2 \* a \* r1 - a

C3 = 2 \* r2

D\_delta = abs(C3 \* delta\_pos[j] - positions[i, j])

X3 = delta\_pos[j] - A3 \* D\_delta

positions[i, j] = (X1 + X2 + X3) / 3

if debug:

print(f"Iteration {iter+1}/{max\_iter} | Optimal Objective Value: {alpha\_score:.6f}")

return alpha\_pos, alpha\_score

if \_\_name\_\_ == "\_\_main\_\_":

start\_total = time.perf\_counter()

print("1. Loading data...")

data = pd.read\_excel('CS.xlsx').dropna()

X = data.iloc[:, 0:9].values

y = data.iloc[:, 9].values

print(f"Data scale: {X.shape[0]} samples × {X.shape[1]} features\n")

print("2. Splitting training set(70%)/test set(30%)...")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y,

test\_size=0.3,

random\_state=0,

shuffle=True

)

print("3. Starting Grey Wolf Optimization (GWO)...")

start\_search = time.perf\_counter()

best\_params\_vec, best\_score = gwo(

func=gwo\_objective,

lb=lb,

ub=ub,

dim=len(lb),

search\_agents=20,

max\_iter=50,

debug=True

)

end\_search = time.perf\_counter()

search\_time = end\_search - start\_search

best\_params = {

'rf\_\_model\_\_n\_estimators': int(round(best\_params\_vec[0])),

'rf\_\_model\_\_max\_depth': int(round(best\_params\_vec[1])),

'svr\_\_model\_\_C': best\_params\_vec[2],

'svr\_\_model\_\_epsilon': best\_params\_vec[3],

'mlp\_\_model\_\_hidden\_layer\_sizes': hlayer\_map[int(round(best\_params\_vec[4]))],

'mlp\_\_model\_\_alpha': best\_params\_vec[5],

'xgb\_\_model\_\_n\_estimators': int(round(best\_params\_vec[6])),

'xgb\_\_model\_\_max\_depth': int(round(best\_params\_vec[7])),

'xgb\_\_model\_\_learning\_rate': best\_params\_vec[8],

'cat\_\_model\_\_iterations': int(round(best\_params\_vec[9])),

'cat\_\_model\_\_depth': int(round(best\_params\_vec[10])),

'cat\_\_model\_\_learning\_rate': best\_params\_vec[11],

'ada\_\_model\_\_n\_estimators': int(round(best\_params\_vec[12])),

'ada\_\_model\_\_learning\_rate': best\_params\_vec[13],

'final\_estimator\_\_fit\_intercept': fit\_intercept\_map[int(round(best\_params\_vec[14]))]

}

print("\nOptimal parameter combination:")

for param, value in best\_params.items():

print(f" {param}: {value}")

estimators = [

('rf', Pipeline([('scaler', MinMaxScaler()), ('model', RandomForestRegressor(random\_state=0, n\_jobs=TREE\_THREADS, max\_features='sqrt'))])),

('svr', Pipeline([('scaler', MinMaxScaler()), ('model', SVR(kernel='rbf', cache\_size=100))])),

('mlp', Pipeline([('scaler', MinMaxScaler()), ('model', MLPRegressor(random\_state=0, early\_stopping=True, max\_iter=300, batch\_size=32))])),

('xgb', Pipeline([('scaler', MinMaxScaler()), ('model', XGBRegressor(random\_state=0, n\_jobs=TREE\_THREADS, subsample=0.8, verbosity=0))])),

('cat', Pipeline([('scaler', MinMaxScaler()), ('model', CatBoostRegressor(random\_state=0, verbose=0, train\_dir='catboost\_info', thread\_count=TREE\_THREADS))])),

('ada', Pipeline([('scaler', MinMaxScaler()), ('model', AdaBoostRegressor(random\_state=0, estimator=DecisionTreeRegressor(max\_depth=3)))]))

]

best\_model = StackingRegressor(

estimators=estimators,

final\_estimator=LinearRegression(),

cv=5, n\_jobs=-1, passthrough=False

).set\_params(\*\*best\_params)

best\_model.fit(X\_train, y\_train)

print("\n4. Conducting prediction...")

start\_predict = time.perf\_counter()

y\_train\_pred = best\_model.predict(X\_train)

y\_test\_pred = best\_model.predict(X\_test)

end\_predict = time.perf\_counter()

predict\_time = end\_predict - start\_predict

train\_path = 'Stacking-GWO-train.xlsx'

pd.DataFrame({'Actual': y\_train, 'Predicted': y\_train\_pred}).to\_excel(train\_path, index=False)

print(f" Training set prediction results saved: {train\_path}")

test\_path = 'Stacking-GWO-test.xlsx'

pd.DataFrame({'Actual': y\_test, 'Predicted': y\_test\_pred}).to\_excel(test\_path, index=False)

print(f" Test set prediction results saved: {test\_path}")

full\_actual = np.concatenate([y\_train, y\_test])

full\_predicted = np.concatenate([y\_train\_pred, y\_test\_pred])

full\_path = 'Stacking-GWO-full.xlsx'

pd.DataFrame({'Actual': full\_actual, 'Predicted': full\_predicted}).to\_excel(full\_path, index=False)

print(f" Full dataset prediction results saved: {full\_path}")

metrics\_train = calculate\_metrics(y\_train, y\_train\_pred)

metrics\_test = calculate\_metrics(y\_test, y\_test\_pred)

metrics\_full = calculate\_metrics(full\_actual, full\_predicted)

total\_time = time.perf\_counter() - start\_total

print("\n==================== Result Summary ====================")

print(f"Time statistics:")

print(f" - Grey Wolf Optimization(GWO): {search\_time:.2f}s ({search\_time/60:.2f}mins)")

print(f" - Prediction time: {predict\_time:.2f}s")

print(f" - Total running time: {total\_time:.2f}s ({total\_time/60:.2f}mins)\n")

print(f"Training set metrics:")

print(f" R²={metrics\_train[0]:.4f}, RMSE={metrics\_train[1]:.4f}, RRMSE={metrics\_train[2]:.4f}, MAE={metrics\_train[3]:.4f}, ρ={metrics\_train[4]:.4f}")

print(f"Test set metrics:")

print(f" R²={metrics\_test[0]:.4f}, RMSE={metrics\_test[1]:.4f}, RRMSE={metrics\_test[2]:.4f}, MAE={metrics\_test[3]:.4f}, ρ={metrics\_test[4]:.4f}")

print(f"Full dataset metrics:")

print(f" R²={metrics\_full[0]:.4f}, RMSE={metrics\_full[1]:.4f}, RRMSE={metrics\_full[2]:.4f}, MAE={metrics\_full[3]:.4f}, ρ={metrics\_full[4]:.4f}")

print("========================================================")

**Code S6 Stacking-TPE model**

pip install hyperopt

import pandas as pd

import numpy as np

import time

import os

import warnings

import random

warnings.filterwarnings('ignore')

from hyperopt import fmin, tpe, hp, STATUS\_OK, Trials

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.ensemble import RandomForestRegressor, AdaBoostRegressor

from sklearn.svm import SVR

from sklearn.neural\_network import MLPRegressor

from sklearn.tree import DecisionTreeRegressor

from sklearn.pipeline import Pipeline

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_squared\_error, r2\_score, mean\_absolute\_error

from sklearn.linear\_model import LinearRegression

from xgboost import XGBRegressor

from catboost import CatBoostRegressor

from sklearn.ensemble import StackingRegressor

os.makedirs('catboost\_info', exist\_ok=True)

TREE\_THREADS = 2

random.seed(0)

np.random.seed(0)

def calculate\_metrics(actual, predicted):

r2 = r2\_score(actual, predicted)

rmse = np.sqrt(mean\_squared\_error(actual, predicted))

rrmse = rmse / np.mean(actual) if np.mean(actual) != 0 else np.inf

mae = mean\_absolute\_error(actual, predicted)

rho = rrmse / (1 + np.sqrt(max(r2, 0.001)))

return r2, rmse, rrmse, mae, rho

hlayer\_map = {0: (30,), 1: (30, 20), 2: (50,)}

fit\_intercept\_map = {0: False, 1: True}

search\_space = {

'rf\_n\_estimators': hp.quniform('rf\_n\_estimators', low=50, high=200, q=1),

'rf\_max\_depth': hp.quniform('rf\_max\_depth', low=4, high=8, q=1),

'svr\_C': hp.uniform('svr\_C', low=1, high=50),

'svr\_epsilon': hp.uniform('svr\_epsilon', low=0.05, high=0.3),

'mlp\_hidden\_idx': hp.quniform('mlp\_hidden\_idx', low=0, high=2, q=1),

'mlp\_alpha': hp.uniform('mlp\_alpha', low=0.001, high=0.01),

'xgb\_n\_estimators': hp.quniform('xgb\_n\_estimators', low=50, high=200, q=1),

'xgb\_max\_depth': hp.quniform('xgb\_max\_depth', low=4, high=6, q=1),

'xgb\_learning\_rate': hp.uniform('xgb\_learning\_rate', low=0.05, high=0.15),

'cat\_iterations': hp.quniform('cat\_iterations', low=50, high=200, q=1),

'cat\_depth': hp.quniform('cat\_depth', low=4, high=6, q=1),

'cat\_learning\_rate': hp.uniform('cat\_learning\_rate', low=0.05, high=0.15),

'ada\_n\_estimators': hp.quniform('ada\_n\_estimators', low=50, high=200, q=1),

'ada\_learning\_rate': hp.uniform('ada\_learning\_rate', low=0.05, high=0.15),

'final\_fit\_intercept\_idx': hp.quniform('final\_fit\_intercept\_idx', low=0, high=1, q=1)

}

def tpe\_objective(params):

try:

parsed\_params = {

'rf\_n\_estimators': int(params['rf\_n\_estimators']),

'rf\_max\_depth': int(params['rf\_max\_depth']),

'svr\_C': params['svr\_C'],

'svr\_epsilon': params['svr\_epsilon'],

'mlp\_hidden': hlayer\_map[int(params['mlp\_hidden\_idx'])],

'mlp\_alpha': params['mlp\_alpha'],

'xgb\_n\_estimators': int(params['xgb\_n\_estimators']),

'xgb\_max\_depth': int(params['xgb\_max\_depth']),

'xgb\_learning\_rate': params['xgb\_learning\_rate'],

'cat\_iterations': int(params['cat\_iterations']),

'cat\_depth': int(params['cat\_depth']),

'cat\_learning\_rate': params['cat\_learning\_rate'],

'ada\_n\_estimators': int(params['ada\_n\_estimators']),

'ada\_learning\_rate': params['ada\_learning\_rate'],

'final\_fit\_intercept': fit\_intercept\_map[int(params['final\_fit\_intercept\_idx'])]

}

estimators = [

('rf', Pipeline([

('scaler', MinMaxScaler()),

('model', RandomForestRegressor(

random\_state=0,

n\_jobs=TREE\_THREADS,

max\_features='sqrt',

n\_estimators=parsed\_params['rf\_n\_estimators'],

max\_depth=parsed\_params['rf\_max\_depth']

))

])),

('svr', Pipeline([

('scaler', MinMaxScaler()),

('model', SVR(

kernel='rbf',

cache\_size=100,

C=parsed\_params['svr\_C'],

epsilon=parsed\_params['svr\_epsilon']

))

])),

('mlp', Pipeline([

('scaler', MinMaxScaler()),

('model', MLPRegressor(

random\_state=0,

early\_stopping=True,

max\_iter=300,

batch\_size=32,

hidden\_layer\_sizes=parsed\_params['mlp\_hidden'],

alpha=parsed\_params['mlp\_alpha']

))

])),

('xgb', Pipeline([

('scaler', MinMaxScaler()),

('model', XGBRegressor(

random\_state=0,

n\_jobs=TREE\_THREADS,

subsample=0.8,

verbosity=0,

n\_estimators=parsed\_params['xgb\_n\_estimators'],

max\_depth=parsed\_params['xgb\_max\_depth'],

learning\_rate=parsed\_params['xgb\_learning\_rate']

))

])),

('cat', Pipeline([

('scaler', MinMaxScaler()),

('model', CatBoostRegressor(

random\_state=0,

verbose=0,

train\_dir='catboost\_info',

thread\_count=TREE\_THREADS,

iterations=parsed\_params['cat\_iterations'],

depth=parsed\_params['cat\_depth'],

learning\_rate=parsed\_params['cat\_learning\_rate']

))

])),

('ada', Pipeline([

('scaler', MinMaxScaler()),

('model', AdaBoostRegressor(

random\_state=0,

estimator=DecisionTreeRegressor(max\_depth=3),

n\_estimators=parsed\_params['ada\_n\_estimators'],

learning\_rate=parsed\_params['ada\_learning\_rate']

))

]))

]

stacking\_model = StackingRegressor(

estimators=estimators,

final\_estimator=LinearRegression(fit\_intercept=parsed\_params['final\_fit\_intercept']),

cv=5,

n\_jobs=-1,

passthrough=False

)

cv\_scores = cross\_val\_score(

stacking\_model, X\_train, y\_train,

cv=5, scoring='r2', n\_jobs=-1

)

mean\_r2 = cv\_scores.mean()

loss = -mean\_r2

print(f"Current parameters: {parsed\_params} | Cross-validation R²: {mean\_r2:.4f} | Loss: {loss:.4f}")

return {'loss': loss, 'status': STATUS\_OK}

except Exception as e:

print(f"Parameter invalid or training failed: {e} | Current parameters: {params}")

return {'loss': 1000.0, 'status': STATUS\_OK}

if \_\_name\_\_ == "\_\_main\_\_":

start\_total = time.perf\_counter()

print("1. Loading data...")

data = pd.read\_excel('CS.xlsx').dropna()

X = data.iloc[:, 0:9].values

y = data.iloc[:, 9].values

print(f"Data scale: {X.shape[0]} samples × {X.shape[1]} features\n")

print("2. Splitting training set(70%)/test set(30%)...")

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y,

test\_size=0.3,

random\_state=0,

shuffle=True

)

print("3. Starting Tree-Structured Parzen Estimator (TPE) optimization...")

start\_search = time.perf\_counter()

trials = Trials()

best\_params\_tpe = fmin(

fn=tpe\_objective,

space=search\_space,

algo=tpe.suggest,

max\_evals=100,

trials=trials,

rstate=np.random.default\_rng(0),

verbose=1

)

end\_search = time.perf\_counter()

search\_time = end\_search - start\_search

best\_params = {

'rf\_\_model\_\_n\_estimators': int(best\_params\_tpe['rf\_n\_estimators']),

'rf\_\_model\_\_max\_depth': int(best\_params\_tpe['rf\_max\_depth']),

'svr\_\_model\_\_C': best\_params\_tpe['svr\_C'],

'svr\_\_model\_\_epsilon': best\_params\_tpe['svr\_epsilon'],

'mlp\_\_model\_\_hidden\_layer\_sizes': hlayer\_map[int(best\_params\_tpe['mlp\_hidden\_idx'])],

'mlp\_\_model\_\_alpha': best\_params\_tpe['mlp\_alpha'],

'xgb\_\_model\_\_n\_estimators': int(best\_params\_tpe['xgb\_n\_estimators']),

'xgb\_\_model\_\_max\_depth': int(best\_params\_tpe['xgb\_max\_depth']),

'xgb\_\_model\_\_learning\_rate': best\_params\_tpe['xgb\_learning\_rate'],

'cat\_\_model\_\_iterations': int(best\_params\_tpe['cat\_iterations']),

'cat\_\_model\_\_depth': int(best\_params\_tpe['cat\_depth']),

'cat\_\_model\_\_learning\_rate': best\_params\_tpe['cat\_learning\_rate'],

'ada\_\_model\_\_n\_estimators': int(best\_params\_tpe['ada\_n\_estimators']),

'ada\_\_model\_\_learning\_rate': best\_params\_tpe['ada\_learning\_rate'],

'final\_estimator\_\_fit\_intercept': fit\_intercept\_map[int(best\_params\_tpe['final\_fit\_intercept\_idx'])]

}

print("\nOptimal parameter combination obtained by TPE optimization:")

for param, value in best\_params.items():

print(f" {param}: {value}")

estimators = [

('rf', Pipeline([('scaler', MinMaxScaler()), ('model', RandomForestRegressor(random\_state=0, n\_jobs=TREE\_THREADS, max\_features='sqrt'))])),

('svr', Pipeline([('scaler', MinMaxScaler()), ('model', SVR(kernel='rbf', cache\_size=100))])),

('mlp', Pipeline([('scaler', MinMaxScaler()), ('model', MLPRegressor(random\_state=0, early\_stopping=True, max\_iter=300, batch\_size=32))])),

('xgb', Pipeline([('scaler', MinMaxScaler()), ('model', XGBRegressor(random\_state=0, n\_jobs=TREE\_THREADS, subsample=0.8, verbosity=0))])),

('cat', Pipeline([('scaler', MinMaxScaler()), ('model', CatBoostRegressor(random\_state=0, verbose=0, train\_dir='catboost\_info', thread\_count=TREE\_THREADS))])),

('ada', Pipeline([('scaler', MinMaxScaler()), ('model', AdaBoostRegressor(random\_state=0, estimator=DecisionTreeRegressor(max\_depth=3)))]))

]

best\_model = StackingRegressor(

estimators=estimators,

final\_estimator=LinearRegression(),

cv=5, n\_jobs=-1, passthrough=False

).set\_params(\*\*best\_params)

best\_model.fit(X\_train, y\_train)

print("\n4. Conducting prediction...")

start\_predict = time.perf\_counter()

y\_train\_pred = best\_model.predict(X\_train)

y\_test\_pred = best\_model.predict(X\_test)

end\_predict = time.perf\_counter()

predict\_time = end\_predict - start\_predict

train\_path = 'Stacking-TPE-train.xlsx'

pd.DataFrame({'Actual': y\_train, 'Predicted': y\_train\_pred}).to\_excel(train\_path, index=False)

print(f" Training set prediction results saved: {train\_path}")

test\_path = 'Stacking-TPE-test.xlsx'

pd.DataFrame({'Actual': y\_test, 'Predicted': y\_test\_pred}).to\_excel(test\_path, index=False)

print(f" Test set prediction results saved: {test\_path}")

full\_actual = np.concatenate([y\_train, y\_test])

full\_predicted = np.concatenate([y\_train\_pred, y\_test\_pred])

full\_path = 'Stacking-TPE-full.xlsx'

pd.DataFrame({'Actual': full\_actual, 'Predicted': full\_predicted}).to\_excel(full\_path, index=False)

print(f" Full dataset prediction results saved: {full\_path}")

metrics\_train = calculate\_metrics(y\_train, y\_train\_pred)

metrics\_test = calculate\_metrics(y\_test, y\_test\_pred)

metrics\_full = calculate\_metrics(full\_actual, full\_predicted)

total\_time = time.perf\_counter() - start\_total

print("\n==================== Result Summary ====================")

print(f"Time statistics:")

print(f" - Tree-Structured Parzen Estimator(TPE) optimization: {search\_time:.2f}s ({search\_time/60:.2f}mins)")

print(f" - Prediction time: {predict\_time:.2f}s")

print(f" - Total running time: {total\_time:.2f}s ({total\_time/60:.2f}mins)\n")

print(f"Training set metrics:")

print(f" R²={metrics\_train[0]:.4f}, RMSE={metrics\_train[1]:.4f}, RRMSE={metrics\_train[2]:.4f}, MAE={metrics\_train[3]:.4f}, ρ={metrics\_train[4]:.4f}")

print(f"Test set metrics:")

print(f" R²={metrics\_test[0]:.4f}, RMSE={metrics\_test[1]:.4f}, RRMSE={metrics\_test[2]:.4f}, MAE={metrics\_test[3]:.4f}, ρ={metrics\_test[4]:.4f}")

print(f"Full dataset metrics:")

print(f" R²={metrics\_full[0]:.4f}, RMSE={metrics\_full[1]:.4f}, RRMSE={metrics\_full[2]:.4f}, MAE={metrics\_full[3]:.4f}, ρ={metrics\_full[4]:.4f}")

print("========================================================")

**Code S7 GUI tool for AALRS strength prediction**

from joblib import dump

dump(best\_model, 'CS-Stacking-model.joblib')

import tkinter as tk

from tkinter import ttk, messagebox, filedialog

import numpy as np

from joblib import load

import warnings

class featureNameSetter:

def \_\_init\_\_(self, model=None, feature\_names=None):

self.model = model

self.feature\_names = feature\_names

def set\_feature\_names(self, feature\_names):

self.feature\_names = feature\_names

predict\_root = tk.Tk()

predict\_root.title("Optimization Prediction Interface")

predict\_root.geometry("1600x800")

warnings.filterwarnings('ignore', category=UserWarning)

style = ttk.Style(predict\_root)

available\_themes = style.theme\_names()

print("Available Themes:", available\_themes)

try:

style.theme\_use('classic')

except:

print("clam theme unavailable, use default theme")

style.configure("TEntry",

font=('Times New Roman', 20, "bold"),

foreground='black',

padding=5)

style.configure("TFrame", borderwidth=5, relief="groove")

style.configure("TLabel", font=('Times New Roman', 16, "bold"))

style.configure("TButton",

background='#0078D4',

foreground='black',

bordercolor='#0078D4',

borderwidth=1,

font=('Times New Roman', 18, "bold"))

style.map('TButton',

background=[('active', '#005A9E'), ('pressed', '#004A8C')],

foreground=[('active', 'white'), ('pressed', 'white')])

def create\_styled\_entry(parent, textvariable=None, width=12):

return tk.Entry(parent,

textvariable=textvariable,

width=width,

font=('Times New Roman', 16, 'bold'),

bg='white',

fg='black',

relief='solid',

bd=1)

def aim\_function(\*\*kwargs):

try:

try:

models = {

'CS': load('CS-Stacking-model.joblib'),

}

except Exception as e:

messagebox.showerror("Model Loading Error", f"Failed to load model file: {str(e)}\nPlease ensure the model file 'CS-Stacking-model.joblib' exists.")

return [0.0]

feature\_mapping = {

'CS': ['MPD', 'Si/Al', 'W/B', 'AC', 'SM', 'CA', 'IT', 'TT', 'VD'],

}

if not hasattr(models['CS'], 'predict'):

raise ValueError("Model file loading failed, please check if the model file is correct.")

if len(feature\_mapping['CS']) != 9:

raise ValueError("Feature mapping length is incorrect, please check if the feature mapping is consistent with the model training.")

cs\_features = [kwargs[var] for var in feature\_mapping['CS']]

if not (0 <= kwargs['MPD'] <= 247.45 and

0.98 <= kwargs['Si/Al'] <= 5.61 and

0.04 <= kwargs['W/B'] <= 0.491 and

0 <= kwargs['AC'] <= 0.155 and

0 <= kwargs['SM'] <= 3.34 and

0 <= kwargs['CA'] <= 108 and

-196 <= kwargs['IT'] <= 120 and

-196 <= kwargs['TT'] <= 600 and

0.2 <= kwargs['VD'] <= 101325):

raise ValueError("Input features are out of reasonable range, please check the input data.")

cs = models['CS'].predict(np.array(cs\_features).reshape(1, -1))[0]

return [cs]

except Exception as e:

messagebox.showerror("Error", f"Model Prediction Error: {str(e)}")

return [0.0]

def update\_predict():

try:

input\_values = {}

for var\_name, var in input\_vars.items():

if var\_name == 'AC':

ac\_value = var.get()

try:

ac\_value = float(ac\_value) / 100

except ValueError:

ac\_value = float(ac\_value)

input\_values[var\_name] = ac\_value

else:

input\_values[var\_name] = float(var.get())

results = aim\_function(\*\*input\_values)

for i, (result\_var, result) in enumerate(zip(output\_vars, results)):

formatted\_result = f"{result:.2f}"

result\_var.delete(0, tk.END)

result\_var.insert(0, formatted\_result)

except ValueError as e:

messagebox.showerror("Error", f"Input Data Format Error: {str(e)}")

except Exception as e:

messagebox.showerror("Error", f"An error occurred during prediction: {str(e)}")

def clear\_entries():

for var in input\_vars.values():

var.set("")

for entry in output\_vars:

entry.delete(0, tk.END)

def save\_entries():

file\_path = filedialog.asksaveasfilename(defaultextension=".txt", filetypes=[("Text Files", "\*.txt"), ("All Files", "\*.\*")])

if file\_path:

with open(file\_path, "w") as file:

input\_data = ""

for var\_name, var in input\_vars.items():

input\_data += f"{var\_name}{input\_units[var\_name]}: {var.get()}\n"

output\_data = ""

for i, entry in enumerate(output\_vars):

output\_data += f"{output\_labels[i]}{output\_units[output\_labels[i]]}: {entry.get()}\n"

file.write(input\_data + "\n" + output\_data)

messagebox.showinfo("Save Successful", "Data has been successfully saved to the file!")

def return\_main\_menu():

predict\_root.destroy()

input\_units = {

'MPD': ' (µm)',

'Si/Al': ' (--)',

'W/B': ' (--)' ,

'AC': ' (%)',

'SM': ' (--)',

'CA': ' (days)',

'IT': ' (°C)',

'TT': ' (°C)',

'VD': ' (Pa)'

}

output\_units = {

'CS': ' (MPa)'

}

title\_label = tk.Label(predict\_root, text="Optimization Prediction Interface", font=('Times New Roman', 24, "bold"))

title\_label.grid(row=0, column=0, columnspan=3, sticky="ew", padx=10, pady=10)

input\_outer\_frame = ttk.Frame(predict\_root, padding="0 0 0 0", style="TFrame")

input\_outer\_frame.grid(row=1, column=0, sticky="nsew", padx=(5, 5))

input\_title\_label = ttk.Label(input\_outer\_frame, text="Input panel", font=('Times New Roman', 18, "bold"))

input\_title\_label.pack(fill="x", padx=300, pady=10)

input\_frame = ttk.Frame(input\_outer\_frame, padding="6 6 6 15")

input\_frame.pack(fill="both", expand=True, padx=6, pady=5)

input\_vars = {}

original\_input\_labels = ['MPD', 'Si/Al', 'W/B', 'AC', 'SM', 'CA', 'IT', 'TT', 'VD']

input\_labels = ['MPD', 'Si/Al', 'W/B', 'AC', 'SM', 'CA', 'IT', 'TT', 'VD']

default\_values = {

'MPD': '38.20',

'Si/Al': '2.32',

'W/B': '0.30',

'AC': '10.00',

'SM': '0',

'CA': '3.00',

'IT': '93.00',

'TT': '99.60',

'VD': '101325'

}

for i, label in enumerate(input\_labels):

row, col = divmod(i, 3)

full\_label = f"{label}{input\_units[label]}"

ttk.Label(input\_frame, text=full\_label).grid(row=row, column=col\*2, padx=7, pady=15, sticky="ew")

input\_vars[label] = tk.StringVar(value=default\_values.get(label, ""))

entry = create\_styled\_entry(input\_frame, textvariable=input\_vars[label], width=13)

entry.grid(row=row, column=col\*2+1, padx=8, pady=30, sticky="ew")

output\_outer\_frame = ttk.Frame(predict\_root, padding="0 0 0 0", style="TFrame")

output\_outer\_frame.grid(row=1, column=2, sticky="nsew", padx=(5, 10))

output\_title\_label = ttk.Label(output\_outer\_frame, text="Output panel", font=('Times New Roman', 18, "bold"))

output\_title\_label.pack(fill="x", padx=140, pady=10)

output\_frame = ttk.Frame(output\_outer\_frame, padding="6 6 6 6", width=180)

output\_frame.pack(fill="both", expand=True, padx=5, pady=5)

output\_outer\_frame.columnconfigure(0, weight=4)

output\_outer\_frame.columnconfigure(1, weight=1)

output\_labels = ['CS']

output\_vars = []

for i, label in enumerate(output\_labels):

full\_label = f"{label}{output\_units[label]}"

ttk.Label(output\_frame, text=full\_label).grid(row=i, column=0, padx=50, pady=100, sticky="ew")

entry = create\_styled\_entry(output\_frame, width=12)

entry.grid(row=i, column=1, padx=20, pady=80, sticky="ew")

output\_vars.append(entry)

control\_panel\_frame = ttk.Frame(predict\_root, padding="0 0 0 0", style="TFrame")

control\_panel\_frame.grid(row=2, column=0, columnspan=3, sticky="ew", padx=10, pady=5)

control\_panel\_title\_label = ttk.Label(control\_panel\_frame, text="Control panel", font=('Times New Roman', 18, "bold"))

control\_panel\_title\_label.pack(fill="x", padx=590, pady=10)

control\_panel\_inner\_frame = ttk.Frame(control\_panel\_frame, padding="0 0 0 0")

control\_panel\_inner\_frame.pack(fill="x", padx=5, pady=5)

button\_predict = ttk.Button(control\_panel\_inner\_frame, text="Predict", command=update\_predict, width=18)

button\_predict.pack(side="left", padx=(55, 12), pady=20, anchor="n")

button\_clear = ttk.Button(control\_panel\_inner\_frame, text="Clear", command=clear\_entries, width=18)

button\_clear.pack(side="left", padx=(55, 12), pady=20, anchor="n")

button\_save = ttk.Button(control\_panel\_inner\_frame, text="Save", command=save\_entries, width=18)

button\_save.pack(side="left", padx=(55, 12), pady=20, anchor="n")

button\_main\_menu = ttk.Button(control\_panel\_inner\_frame, text="Exit", command=return\_main\_menu, width=18)

button\_main\_menu.pack(side="left", padx=(55, 12), pady=20, anchor="n")

def check\_entry\_style():

test\_entry = ttk.Entry(predict\_root)

print("TEntry Style Options:", style.configure("TEntry"))

test\_entry.destroy()

test\_tk\_entry = tk.Entry(predict\_root, font=('Times New Roman', 20, 'bold'))

print("tk.Entry Font:", test\_tk\_entry.cget('font'))

test\_tk\_entry.destroy()

check\_entry\_style()

predict\_root.mainloop()