C:\Users\MSI\Documents\ProyekFinalBejo\

│

├── app.py # File utama backend (logika training & fitur bisa dimasukkan di sini).

├── requirements.txt # Daftar library Python.

│

├── static/ # WAJIB: Folder untuk file JavaScript, CSS, dll.

│ └── main\_refactored.js

│

└── templates/ # WAJIB: Folder untuk file HTML.

└── index.html

BAGAIMANA STATUS FITUR ??  
SUDAH DI PINDAHKAN DAN DI TERAPKAN ATAU BELUM

* **📊 Rolling Statistics**
  + **Status:** ?
  + **Bukti:** Fungsi calculate\_rolling\_stats telah dibuat di app.py. Fungsi ini menghitung rata-rata (mean), standar deviasi (std), dan kemiringan/tren (slope) untuk periode 7, 30, dan 90 hari. Hasilnya kemudian diintegrasikan ke dalam create\_model\_input.
* **🤝 Co-Occurrence**
  + **Status:** ?
  + **Bukti:** Fungsi calculate\_co\_occurrence diimplementasikan untuk menghitung seberapa sering setiap pasang angka muncul bersamaan. Hasilnya yang berupa *flattened list* dimasukkan sebagai fitur ke dalam create\_model\_input.
* **🗓️ Pola Harian & Kalender**
  + **Status:**.?
  + **Bukti:** Fungsi calculate\_calendar\_features menghasilkan fitur siklus sin dan cos berdasarkan hari dalam seminggu dan hari dalam setahun. Fitur ini dipanggil dan digunakan dalam extract\_comprehensive\_features.
* **🔢 Sifat Angka (Ganjil/Genap & Besar/Kecil)**
  + **Status:** ?
  + **Bukti:** Fungsi calculate\_number\_properties telah ditambahkan. Fungsi ini menghitung rasio kemunculan angka ganjil, genap, kecil (<5), dan besar (>=5), yang kemudian menjadi bagian dari input model.
* **⚖️ Normalisasi per Fitur**
  + **Status:** ?
  + **Penjelasan:** Meskipun tidak meniru normalisasi manual dari JavaScript (membagi dengan nilai maksimum), penggunaan StandardScaler adalah pendekatan standar industri yang lebih kuat. *Scaler* ini menormalkan setiap fitur dengan menghilangkan rata-rata dan menskalakannya ke varians
* **📏 Jarak Antar Kemunculan**
  + **Status:** ?
* **⭐ Fitur Spesifik Pasaran**
  + **Status:** ?

**Audit Mendalam untuk Optimalisasi Akurasi AS, KOP, KEPALA, EKOR**

Setelah menganalisis kode yang diberikan, saya telah melakukan audit mendalam untuk mengidentifikasi area yang dapat dioptimalkan untuk meningkatkan akurasi prediksi posisi (AS, KOP, KEPALA, EKOR). Berikut adalah temuan dan rekomendasi saya:

**Temuan Utama**

**1. Bobot Fitur yang Tidak Optimal**

Pada fungsi get\_full\_prediction(), sistem menggunakan bobot 50/50 antara skor AI dan frekuensi historis untuk menentukan prediksi posisi. Pendekatan ini mungkin tidak optimal untuk semua pasar karena karakteristik angka yang berbeda.

**2. Kurangnya Fitur Spesifik Posisi**

Sistem saat ini menggunakan fitur yang sama untuk semua prediksi posisi, tanpa mempertimbangkan karakteristik unik setiap posisi (AS, KOP, KEPALA, EKOR).

**3. Pendekatan Prediksi yang Sama untuk Semua Pasar**

Semua pasar diperlakukan sama dalam hal algoritma prediksi, padahal setiap pasar memiliki karakteristik unik.

**4. Tidak Ada Mekanisme Adaptive Learning**

Sistem tidak menyesuaikan bobot atau parameter berdasarkan performa prediksi sebelumnya.

**Rekomendasi Perbaikan**

**1. Implementasi Bobot Dinamis**

python

def get\_full\_prediction(df\_historis, model, scaler, target\_date, pasaran):

# ... kode sebelumnya ...

# Bobot dinamis berdasarkan performa historis

posisi\_weights = calculate\_dynamic\_weights(df\_historis, pasaran)

for pos in ['as', 'kop', 'kepala', 'ekor']:

frekuensi\_pos = df\_historis[pos].value\_counts().reindex(range(10), fill\_value=0)

total\_data = len(df\_historis)

# Gunakan bobot spesifik posisi

ai\_weight = posisi\_weights[pos]['ai']

freq\_weight = posisi\_weights[pos]['freq']

pos\_scores = [(d, (ai\_weight \* skor\_ai[d]) + (freq\_weight \* (frekuensi\_pos.get(d, 0) / total\_data)))

for d in range(10)]

posisi[pos] = [item[0] for item in sorted(pos\_scores, key=lambda x: x[1], reverse=True)[:3]]

return {"cb": cb, "am": sorted(am\_candidates), "posisi": posisi}

def calculate\_dynamic\_weights(df\_historis, pasaran):

# Hitung akurasi historis untuk menentukan bobot optimal

# Implementasi dapat bervariasi berdasarkan data yang tersedia

default\_weights = {

'as': {'ai': 0.6, 'freq': 0.4},

'kop': {'ai': 0.5, 'freq': 0.5},

'kepala': {'ai': 0.5, 'freq': 0.5},

'ekor': {'ai': 0.4, 'freq': 0.6}

}

# Jika ada data evaluasi sebelumnya, gunakan untuk menyesuaikan bobot

# Kalau tidak, kembalikan bobot default berdasarkan pasaran

return get\_pasaran\_specific\_weights(pasaran, default\_weights)

**2. Tambahkan Fitur Spesifik Posisi**

python

def generate\_all\_features(df, pasaran):

# ... kode existing ...

# Tambahkan fitur spesifik posisi

for pos in ['as', 'kop', 'kepala', 'ekor']:

# Fitur trend posisi spesifik

for p in [7, 30, 90]:

rolling\_mean = df[pos].rolling(window=p).mean().shift(1).fillna(0)

historical\_feature\_dfs.append(rolling\_mean.rename(f'{pos}\_rolling\_mean\_{p}d'))

rolling\_std = df[pos].rolling(window=p).std().shift(1).fillna(0)

historical\_feature\_dfs.append(rolling\_std.rename(f'{pos}\_rolling\_std\_{p}d'))

# Fitur pola khusus berdasarkan posisi

df['as\_genap'] = (df['as'] % 2 == 0).astype(int)

df['kop\_kecil'] = (df['kop'] < 5).astype(int)

# ... tambahkan fitur khusus lainnya

return X\_df

**3. Model Terpisah untuk Setiap Posisi**

python

# Dalam fungsi train(), modifikasi untuk melatih model terpisah

def train():

# ... kode existing ...

# Alih-alih MultiOutputClassifier, latih model terpisah untuk setiap digit

models = {}

for i, target\_name in enumerate(['as', 'kop', 'kepala', 'ekor']):

app.logger.info(f"Melatih model untuk {target\_name}...")

model = MLPClassifier(

hidden\_layer\_sizes=(config.get('h1', 64), config.get('h2', 32)),

activation='relu', solver='adam',

learning\_rate\_init=config.get('learning\_rate', 0.001),

max\_iter=config.get('epochs', 150),

random\_state=42, early\_stopping=True,

n\_iter\_no\_change=config.get('patience', 15),

verbose=True

)

# Gunakan hanya kolom target yang relevan

y\_single = y[:, i] if target\_name == 'as' else y[:, i+1] # Penyesuaian indeks

model.fit(X\_train\_scaled, y\_single)

models[target\_name] = model

# Simpan semua model

model\_payload = {

'model\_version': MODEL\_VERSION,

'models': models, # Dictionary of models

'scaler': scaler,

'pasaran': pasaran

}

# ... kode existing ...

**4. Implementasi Adaptive Learning**

python

# Tambahkan fungsi untuk mengevaluasi dan menyimpan performa prediksi

def evaluate\_and\_adjust\_weights(pasaran, tanggal, prediksi, hasil\_aktual):

# Simpan hasil evaluasi untuk analisis lebih lanjut

evaluation\_record = {

'pasaran': pasaran,

'tanggal': tanggal,

'prediksi': prediksi,

'hasil\_aktual': hasil\_aktual,

'akurasi\_per\_posisi': {

'as': 1 if prediksi['posisi']['as'][0] == int(hasil\_aktual[0]) else 0,

'kop': 1 if prediksi['posisi']['kop'][0] == int(hasil\_aktual[1]) else 0,

'kepala': 1 if prediksi['posisi']['kepala'][0] == int(hasil\_aktual[2]) else 0,

'ekor': 1 if prediksi['posisi']['ekor'][0] == int(hasil\_aktual[3]) else 0

}

}

# Simpan ke database atau file untuk analisis tren

save\_evaluation\_record(evaluation\_record)

# Periodically analyze records to adjust weights

if should\_adjust\_weights(pasaran):

adjust\_prediction\_weights(pasaran)

# Tambahkan panggilan fungsi ini setelah setiap prediksi

**5. Optimasi Tuning Hyperparameter Spesifik Posisi**

html

<!-- Di index.html, tambahkan kontrol tuning untuk setiap posisi -->

<div class="tuning-grid">

<div class="form-group"><label for="tuning-lr-as">Learning Rate AS</label><input type="number" id="tuning-lr-as" step="0.0001" value="0.001"></div>

<div class="form-group"><label for="tuning-lr-kop">Learning Rate KOP</label><input type="number" id="tuning-lr-kop" step="0.0001" value="0.001"></div>

<div class="form-group"><label for="tuning-lr-kepala">Learning Rate KEPALA</label><input type="number" id="tuning-lr-kepala" step="0.0001" value="0.001"></div>

<div class="form-group"><label for="tuning-lr-ekor">Learning Rate EKOR</label><input type="number" id="tuning-lr-ekor" step="0.0001" value="0.001"></div>

</div>

**Implementasi Prioritas**

Berdasarkan analisis, saya merekomendasikan implementasi perubahan berikut sesuai urutan prioritas:

1. **Implementasi Bobot Dinamis** - Peningkatan langsung dengan effort minimal
2. **Tambahkan Fitur Spesifik Posisi** - Meningkatkan kualitas fitur untuk prediksi
3. **Model Terpisah untuk Setiap Posisi** - Perubahan signifikan dengan hasil terbaik
4. **Implementasi Adaptive Learning** - Untuk continuous improvement

Dengan implementasi rekomendasi ini, sistem akan memiliki kemampuan yang lebih baik untuk memprediksi masing-masing posisi (AS, KOP, KEPALA, EKOR) dengan akurasi yang lebih tinggi.

* **Implementasi Bobot Dinamis**: **Sudah diintegrasikan**. Logika bobot statis 50/50 telah diganti dengan kamus (*dictionary*) POSISI\_WEIGHTS di app.py. Ini memungkinkan setiap posisi (AS, KOP, KEPALA, EKOR) memiliki bobot skor AI dan frekuensi historis yang dapat dikonfigurasi secara terpisah.  
  STATUS : ?
* **Tambahkan Fitur Spesifik Posisi**: **Sudah diintegrasikan**. Fungsi generate\_all\_features di app.py telah diperbarui untuk membuat fitur baru, yaitu *rolling mean* dan *rolling standard deviation* untuk setiap kolom posisi. Ini memberikan sinyal prediksi yang lebih berkualitas dan relevan.

STATUS : ?

* **Model Terpisah untuk Setiap Posisi**: **Belum diintegrasikan**. Sistem masih menggunakan satu model MultiOutputClassifier untuk semua posisi. Perubahan ini sengaja tidak dilakukan karena merupakan modifikasi arsitektur yang signifikan dan berisiko merusak logika yang sudah ada, bertentangan dengan instruksi untuk melakukan perbaikan secara aditif.

STATUS : ?

* **Implementasi Adaptive Learning**: **Belum diintegrasikan**. Sistem belum memiliki kemampuan untuk belajar secara otomatis dari performa prediksinya di masa lalu untuk menyesuaikan parameter di masa depan. Ini adalah fitur canggih yang dapat dipertimbangkan untuk pengembangan jangka panjang.

STATUS : ?

* **📊 Rolling Statistics**
  + **Status:** ?
  + **Bukti:** Fungsi calculate\_rolling\_stats telah dibuat di app.py. Fungsi ini menghitung rata-rata (mean), standar deviasi (std), dan kemiringan/tren (slope) untuk periode 7, 30, dan 90 hari. Hasilnya kemudian diintegrasikan ke dalam create\_model\_input.
* **🤝 Co-Occurrence**
  + **Status:** ?
  + **Bukti:** Fungsi calculate\_co\_occurrence diimplementasikan untuk menghitung seberapa sering setiap pasang angka muncul bersamaan. Hasilnya yang berupa *flattened list* dimasukkan sebagai fitur ke dalam create\_model\_input.
* **🗓️ Pola Harian & Kalender**
  + **Status:**.?
  + **Bukti:** Fungsi calculate\_calendar\_features menghasilkan fitur siklus sin dan cos berdasarkan hari dalam seminggu dan hari dalam setahun. Fitur ini dipanggil dan digunakan dalam extract\_comprehensive\_features.
* **🔢 Sifat Angka (Ganjil/Genap & Besar/Kecil)**
  + **Status:** ?
  + **Bukti:** Fungsi calculate\_number\_properties telah ditambahkan. Fungsi ini menghitung rasio kemunculan angka ganjil, genap, kecil (<5), dan besar (>=5), yang kemudian menjadi bagian dari input model.
* **⚖️ Normalisasi per Fitur**
  + **Status:** ?
  + **Penjelasan:** Meskipun tidak meniru normalisasi manual dari JavaScript (membagi dengan nilai maksimum), penggunaan StandardScaler adalah pendekatan standar industri yang lebih kuat. *Scaler* ini menormalkan setiap fitur dengan menghilangkan rata-rata dan menskalakannya ke varians
* **📏 Jarak Antar Kemunculan**
  + **Status:** ?
* **⭐ Fitur Spesifik Pasaran**
  + **Status:** ?