python
Copy

```
# xenopoulos_algorithm.py
import numpy as np
import tensorflow as tf
import pandas as pd
import matplotlib.pyplot as plt
import requests
import qiskit
from pathlib import Path
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.layers import Dense, Dropout, LSTM
from sklearn.model_selection import train_test_split

class XenopoulosCore:
    """Βασικές λειτουργίες διαλεκτικής σύνθεσης / Core dialectical synthesis operations"""
    def __init__(self):
        self.scaler = MinMaxScaler()

    def _dialectical_operator(self, F, G):
        """Διαλεκτικός τελεστής N = F⊗G / Dialectical operator N = F⊗G"""
        return F * (1 - G**2) + 0.1 * np.exp(-3 * G)

class EconomicDialectics(XenopoulosCore):
    """Διαλεκτική ανάλυση οικονομικών δεικτών / Economic indicators analysis"""
    def __init__(self, country_code='GR'):
        super().__init__()
        self.country_code = country_code
        self.data_file = Path('economic_data.csv')
        self.model = self._build_economic_model()

    def _build_economic_model(self):
        """Μοντέλο LSTM για οικονομικές προβλέψεις / LSTM model for economic predictions"""
        model = tf.keras.Sequential([
            LSTM(128, activation='tanh', input_shape=(5, 1)),
            Dropout(0.3),
            Dense(64, activation='relu'),
            Dense(1, activation='linear')
        ])
        model.compile(optimizer='adam', loss='mse', metrics=['mae'])
        return model

    def fetch_data_from_api(self):
        """Ανάκτηση δεδομένων από Παγκόσμια Τράπεζα / Fetch data from World Bank API"""
        indicators = {
            'NY.GDP.MKTP.KD.ZG': 'GDP_growth',
            'GC.DOD.TOTL.GD.ZS': 'Public_debt',
            'FP.CPI.TOTL.ZG': 'Inflation'
        }

        data = []
        for code, name in indicators.items():
            url = f"http://api.worldbank.org/v2/country/{self.country_code}/indicator/{code}?date=2000:2020&format=json"
            try:
                response = requests.get(url, timeout=15)
                if response.status_code == 200:
                    df = pd.DataFrame(response.json()[1])[['date', 'value']].rename(columns={'value': name}).set_index('date')
                    data.append(df)
            except requests.exceptions.RequestException as e:
                print(f"Σφάλμα API: {e} / API Error: {e}")
                continue
        combined_data = pd.concat(data, axis=1).dropna()
        combined_data.to_csv(self.data_file, index=False)
        return combined_data

    def load_data(self):
        """Φόρτωση δεδομένων από αρχείο ή API / Load data from file or API"""
        if self.data_file.exists():
            return pd.read_csv(self.data_file)
        return self.fetch_data_from_api()

    def visualize_predictions(self, y_true, y_pred):
        """Οπτικοποίηση προβλέψεων / Visualization of predictions"""
        plt.figure(figsize=(10, 6))
        plt.plot(y_true, label='Πραγματικά Δεδομένα/Real Data', color='blue', marker='o')
        plt.plot(y_pred, label='Προβλέψεις/Predictions', color='red', linestyle='--')
        plt.title('Σύγκριση Πραγματικών vs Προβλέψεων / Real vs Predicted Data')
        plt.xlabel('Χρονική Περίοδος/Time Period')
        plt.ylabel('Τιμές Δεικτών/Indicator Values')
        plt.legend()
        plt.grid(True)
        plt.savefig('economic_predictions.png', dpi=300)
        plt.close()

class QuantumDialectics(XenopoulosCore):
    """Πρόβλεψη σφαλμάτων σε κβαντικά συστήματα / Quantum error prediction"""
    def __init__(self, qubit_type='superconducting'):
        super().__init__()
        self.qubit_type = qubit_type
        self.model = self._build_quantum_model()

    def _build_quantum_model(self):
        """Μοντέλο πρόβλεψης σφαλμάτων / Error prediction model"""
        model = tf.keras.Sequential([
            Dense(128, activation='relu', input_shape=(3,)),
            Dropout(0.2),
            Dense(64, activation='relu'),
            Dense(1, activation='sigmoid')
        ])
        model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
        return model

    def predict_error_rate(self, qubit_data):
        """Πρόβλεψη ποσοστού σφαλμάτων / Error rate prediction"""
        scaled_data = self.scaler.fit_transform(np.array(list(qubit_data.values())).T)
        F = scaled_data[:,0]
        G = scaled_data[:,1]
        N = self._dialectical_operator(F, G)
        X = np.column_stack([F, G, N])
        return self.model.predict(X)

    def visualize_errors(self, qubit_params, predictions):
        """Οπτικοποίηση αποτελεσμάτων / Results visualization"""
        plt.figure(figsize=(10, 6))
        plt.plot(qubit_params['CoherenceTime'], predictions, 's--', color='green')
        plt.title('Ποσοστό Σφαλμάτων ανά Χρόνο Συνόχης / Error Rate vs Coherence Time')
        plt.xlabel('Χρόνος Συνόχης (μs)/Coherence Time (μs)')
        plt.ylabel('Ποσοστό Σφαλμάτων (%)/Error Rate (%)')
        plt.grid(True)
        plt.savefig('quantum_errors.png', dpi=300)
        plt.close()

class XenopoulosDialectics:
    """Ενοποιημένη ανάλυση / Unified analysis"""
    def __init__(self, country_code='GR', qubit_type='superconducting'):
        self.economy = EconomicDialectics(country_code)
        self.quantum = QuantumDialectics(qubit_type)

    def run_analysis(self):
        """Εκτέλεση πλήρους ανάλυσης / Run full analysis"""
        economic_data = self.economy.load_data()

        if 'GDP_growth' not in economic_data.columns:
            raise ValueError("Λείπει η στήλη 'GDP_growth'/Missing 'GDP_growth' column")

        X = economic_data.drop(columns=['GDP_growth']).values
        y = economic_data['GDP_growth'].values

        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)

        # Διόρθωση διαστάσεων για LSTM / Fix dimensions for LSTM
        X_train = X_train.reshape((X_train.shape[0], X_train.shape[1], 1))
        X_test = X_test.reshape((X_test.shape[0], X_test.shape[1], 1))

        self.economy.model.fit(X_train, y_train, epochs=100, verbose=0)
        predictions = self.economy.model.predict(X_test)
        self.economy.visualize_predictions(y_test, predictions)

        qubit_data = {
            "CoherenceTime": [50, 90, 150, 200],
            "Noise": [8.0, 4.0, 1.0, 0.5]
        }
        error_prediction = self.quantum.predict_error_rate(qubit_data)
        self.quantum.visualize_errors(qubit_data, error_prediction)
        return economic_data, error_prediction

def ethical_constraint(predictions):
    """Ηθικοί περιορισμοί / Ethical constraints"""
    return np.where(predictions < 0, 0, predictions)

if __name__ == "__main__":
    analysis = XenopoulosDialectics()
    economic_data, quantum_errors = analysis.run_analysis()
    final_prediction = ethical_constraint(quantum_errors)
    print(f"Βελτιωμένη Πρόβλεψη Σφαλμάτων/Improved Error Prediction: {final_prediction[0][0]:.2f}
%")
```

text

Copy

```
# requirements.txt
numpy==1.26.4
pandas==2.2.1
tensorflow==2.15.0
scikit-learn==1.4.0
matplotlib==3.8.2
requests==2.31.0
qiskit==1.0.0
```