# Machine Learning Model Details

## Primary Model: Random Forest Regressor

The Mental Fatigue Detector uses a Random Forest Regressor as its primary prediction model. This choice was made for several key reasons:

### Model Configuration

train_with_real_data.py

Copy

```
rf_model = RandomForestRegressor(
    n_estimators=100,
    max_depth=10,
    min_samples_split=5,
    min_samples_leaf=2,
    random_state=42,
    n_jobs=-1
)
```

### Training Data

- **410 real samples**: The model is trained on genuine behavioral data rather than synthetic data
- **Data sources**:
    - Keyboard data (100 samples)
    - Mouse data (22 sessions expanded to ~110 samples)
    - Facial data (200 samples selected from 2,900 facial images)

### Feature Set (14 behavioral indicators)

1. typing_speed
2. error_rate
3. pause_frequency
4. key_press_duration
5. movement_speed
6. click_frequency
7. eye_blink_rate
8. eye_closure_duration
9. speech_rate
10. pitch_variation
11. volume
12. clarity

13. hour_of_day
14. day_of_week

### Feature Importance

The top 5 most predictive features:

1. eye_closure_duration: 43.65%
2. click_frequency: 26.61%
3. eye_blink_rate: 9.72%
4. movement_speed: 3.79%
5. typing_speed: 2.40%

### Performance Metrics

- **Training accuracy**: 89% ($R^2$ score of 0.890)
- **Test accuracy**: 76.6% ($R^2$ score of 0.766)
- **Training MSE(Mean Squared Error)**: 0.0084
- **Test MSE**: 0.0170

### Advantages of Random Forest

1. **Interpretability**: Provides feature importance analysis
2. **Robustness**: Less prone to overfitting than neural networks
3. **Efficiency**: Works well with the available dataset size
4. **Handles mixed data types**: Can process various behavioral indicators

## Fallback Model: Neural Network (4-layer deep network)

A neural network serves as a fallback when the Random Forest model has low confidence:

### Architecture

- **Input layer**: 14 nodes (one for each feature)
- **Hidden layer 1**: 64 neurons with ReLU activation and BatchNormalization
- **Dropout layer**: 30% dropout for regularization
- **Hidden layer 2**: 32 neurons with ReLU activation and BatchNormalization
- **Dropout layer**: 20% dropout
- **Hidden layer 3**: 16 neurons with ReLU activation
- **Output layer**: 1 neuron with linear activation (regression output)

### When It's Used

The system falls back to this neural network when:

1. The Random Forest model has low confidence (below a threshold)
2. There are data quality issues with certain modalities
3. The system detects unusual patterns that might be outside the Random Forest's training distribution

### Training Process

Both models are trained using the same dataset but with different validation approaches:

- Random Forest: Standard train-test split (80/20)
- Neural Network: K-fold cross-validation for better generalization

# Model Integration in the Application

The models are integrated into the application through the

`fatigue_predictor.py`
module:

1. **Data preprocessing**: Raw behavioral data is normalized using StandardScaler
2. **Feature extraction**: The system extracts the 14 behavioral indicators
3. **Prediction**: The Random Forest model predicts the fatigue score
4. **Confidence calculation**: The system calculates confidence based on data quality
5. **Fallback mechanism**: If confidence is low, the Neural Network is used
6. **Result combination**: The final prediction combines individual scores with confidence weights

This comprehensive ML approach ensures reliable fatigue detection across different user behaviors and data quality scenarios.

**Important Files in the Mental Fatigue Detector Project**

**Core Application Files**

1. **fatique/views.py**
   - Contains the main view functions for rendering pages and handling user interactions
   - Includes the dashboard view, data collection endpoints, and analysis functions
2. **fatique/models.py**
   - Defines database models for storing user profiles, behavioral data, and analysis results
   - Contains UserProfile, BehavioralData, KeyboardMetrics, MouseMetrics, FacialMetrics, VoiceMetrics, FatigueAnalysis, ProductivityRecommendation, and ProductivitySession models
3. **fatique/urls.py**
   - Maps URL patterns to view functions

- Includes routes for the dashboard, tracking, fatigue analysis, and recommendations

4. **fatique/admin.py**
   - Configures the Django admin interface for managing application data
   - Registers models with custom admin classes for better organization

**Machine Learning Components**

5. **fatique/ml_models/fatigue_detector.py**
   - Implements the FatigueDetector class that handles prediction of fatigue levels
   - Contains methods for feature extraction, model loading, and prediction
6. **train_with_real_data.py**
   - Script for training the Random Forest model with real datasets
   - Includes data preparation, model training, evaluation, and saving
7. **fatique/ml_models/recommendation_engine.py**
   - Generates personalized productivity recommendations based on fatigue analysis
   - Contains logic for selecting appropriate recommendations by fatigue level
8. **fatique/ml_models/data_preprocessor.py**
   - Handles preprocessing of raw behavioral data
   - Implements feature extraction and normalization using StandardScaler

**API Components**

9. **fatique/api/views.py**
   - Contains API endpoints for the REST interface
   - Includes viewsets for user profiles, behavioral data, and fatigue analysis
10. **fatique/api/serializers.py**
    - Defines serializers for converting model instances to JSON
    - Contains serializers for all models used in the API
11. **fatique/api/urls.py**
    - Configures URL routing for API endpoints
    - Sets up the DefaultRouter for REST framework viewsets

**Data Collection and Integration**

12. **fatique/data_collection/data_collector.py**
    - Implements the DataCollector class for gathering behavioral data
    - Contains methods for collecting keyboard, mouse, and facial data
13. **fatique/datasets/data_integrator.py**
    - Handles integration of different data sources
    - Used by the management command for dataset integration
14. **fatique/management/commands/integrate_datasets.py**
    - Django management command for integrating facial, mouse, and keyboard datasets
    - Used for initial setup and data preparation

**Frontend Templates**

15. **templates/index.html**
    - Main landing page template
    - Introduces the application and its features
16. **templates/dashboard/index.html**
    - Dashboard template with visualizations and analytics
    - Contains Chart.js integration for fatigue trends and component analysis
17. **templates/data_collection.html**
    - Template for the data collection flow
    - Includes JavaScript for capturing keyboard, mouse, and facial data

**Configuration and Project Setup**

18. **mental_fatique/settings.py**
    - Django settings file with configuration for the entire project
    - Includes database settings, installed apps, middleware, and security settings
19. **mental_fatique/urls.py**
    - Main URL configuration for the project
    - Includes routes for the admin interface, API, and main application
20. **requirements.txt**
    - Lists all Python dependencies required for the project
    - Includes Django, scikit-learn, TensorFlow, OpenCV, and other libraries

**Static Assets and Resources**

21. **static/js/data_collection.js**
    - JavaScript for the data collection process
    - Implements event listeners for keyboard and mouse tracking
22. **static/js/dashboard.js**
    - JavaScript for the dashboard functionality
    - Handles chart initialization and data updates
23. **static/css/main.css**
    - Main stylesheet for the application
    - Defines the visual style and responsive design

**Trained Models and Data**

24. **ml_models/trained_models/fatigue_model_real_data.joblib**
    - Saved Random Forest model trained on real data
    - Used for production predictions
25. **ml_models/trained_models/feature_scaler_real_data.joblib**
    - Saved StandardScaler for feature normalization
    - Ensures consistent scaling between training and prediction