



FocusFlow

AI-Powered Productivity Tracking & Prediction System



Complete Project Documentation for Presentation

🎯 Project Objective

Primary Goal: Build an intelligent productivity tracking application that automatically monitors computer usage, classifies activities, and uses Machine Learning to predict future productivity patterns.

Specific Objectives

1. **Automatic Activity Tracking:** Build a background tracker that monitors active windows every 10 seconds without manual input
2. **Smart Classification:** Automatically categorize apps as productive (VS Code, Excel) or distracting (YouTube, Instagram)
3. **Time Series Forecasting:** Implement three ML models (LSTM, ARIMA, Prophet) to predict 7-day productivity

4. **Ensemble Learning:** Combine multiple models using weighted averaging for improved accuracy
5. **Real-time Dashboard:** Display tracked data, analytics, and ML predictions in a beautiful UI
6. **Actionable Insights:** Provide best focus windows, distraction triggers, and behavioral patterns

Why This Project?

- Demonstrates **end-to-end ML pipeline** from data collection to prediction
- Practical application of **time series forecasting** techniques
- Integration of **deep learning (LSTM)** with statistical models (ARIMA, Prophet)
- Full-stack development with **React + Flask + MongoDB**

❓ Problem Statement

The Digital Distraction Crisis

In today's digital world, knowledge workers face constant distractions that significantly impact productivity. Studies show alarming statistics:

11 min

Avg time before
distraction

25 min

Time to refocus

2.5 hrs

Daily time lost

67%

Workers struggle
with focus

Problems with Existing Solutions

ISSUE	TRADITIONAL TOOLS	FOCUSFLOW SOLUTION
Tracking Method	✗ Manual time entry (tedious)	✓ Automatic background tracking
Analysis Type	✗ Basic statistics only	✓ ML-powered predictions
Future Planning	✗ No forecasting capability	✓ 7-day productivity forecasts
AI/ML Integration	✗ None	✓ LSTM + ARIMA + Prophet ensemble
Personalization	✗ Generic advice	✓ Pattern-based recommendations



Methodology

Overall Approach

We follow a systematic data science methodology with these phases:

Data Collection → Preprocessing → Feature Engineering → Model Training → Evaluation → Deployment → Visualization

Step 1: Data Collection (Activity Tracking)

A Python background process monitors the active window every 10 seconds:

```
# Simplified tracking logic (tracker.py) while tracking:  
active_window = get_active_window() # pygetwindow library app_name =  
active_window.title # e.g., "VS Code - main.py" category =  
categorize_app(app_name) # productive/distracting/neutral  
log_activity(app_name, category, 10) # Save to MongoDB  
time.sleep(10) # Wait 10 seconds
```

Step 2: Data Storage Schema

```
# MongoDB Activity Document Structure { "user_id": "ObjectId(...)",  
"app_name": "VS Code - project.py", "category": "productive",  
"is_productive": true, "duration_minutes": 0.167, # 10 seconds =  
0.167 min "timestamp": "2024-01-15T10:30:00Z" }
```

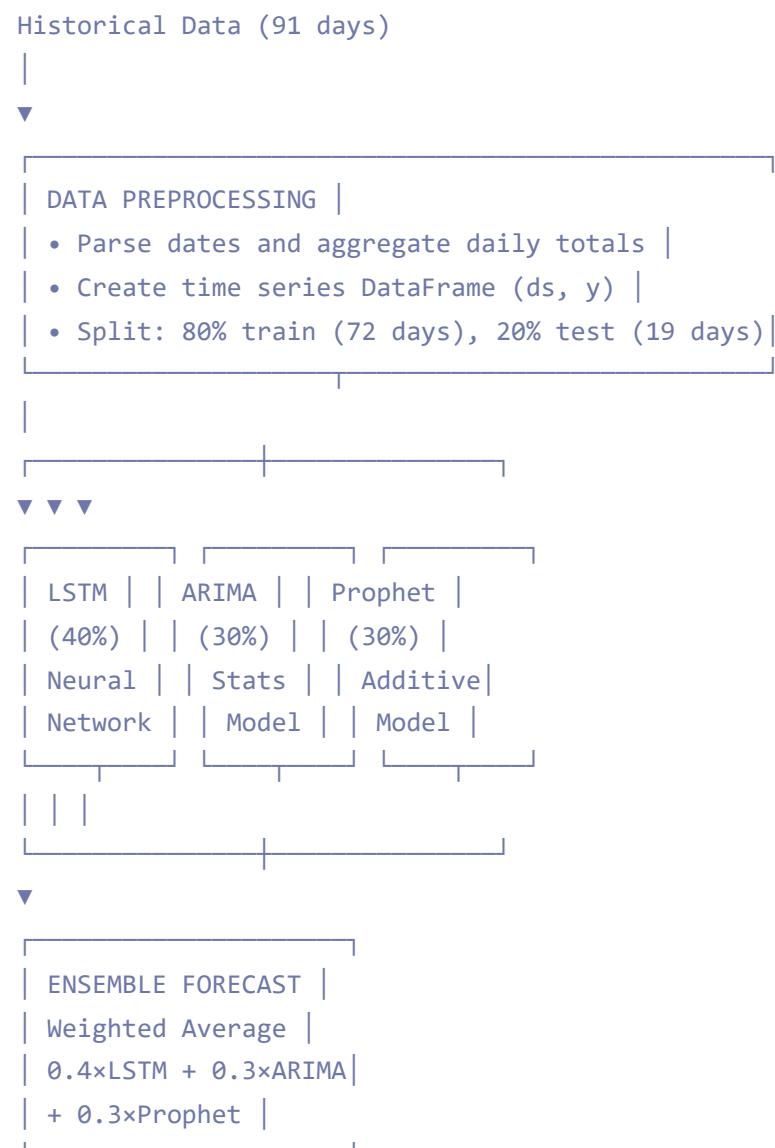
Step 3: Data Preprocessing

- **Aggregation:** Sum daily productive minutes from raw activities
- **Time Series Format:** Convert to Prophet-compatible format (ds, y columns)
- **Train/Test Split:** 80% training (72 days), 20% testing (19 days)
- **Normalization:** MinMaxScaler for LSTM input (0-1 range)

Step 4: Feature Engineering

```
# Features extracted for productivity classification features = {  
    'tasks_completed': 15, # From task management 'completion_rate':  
    75.0, # Percentage 'productive_time': 1200, # Minutes this week  
    'distraction_time': 400, # Minutes this week 'focus_sessions': 12, #  
    Pomodoro-style sessions 'avg_session_duration': 45, # Minutes  
    'consistency_score': 85.7, # Days with >2hrs productive  
    'focus_ratio': 0.75 # productive / (productive + distracted) }
```

Step 5: Model Training Pipeline



Step 6: Evaluation Metrics

- **MAE (Mean Absolute Error)**: Average prediction error in minutes
- **RMSE (Root Mean Square Error)**: Penalizes larger errors more heavily
- **MAPE (Mean Absolute Percentage Error)**: Error as a percentage of actual value

E System Architecture

-- FRONTEND (REACT + TYPESCRIPT) --

Dashboard

Real-time stats

Analytics

Charts & trends

ML Insights

Predictions

Task Manager

CRUD tasks

▼ HTTP/REST API ▼

-- BACKEND (FLASK + PYTHON) --

Auth Routes

JWT tokens

Activity Routes

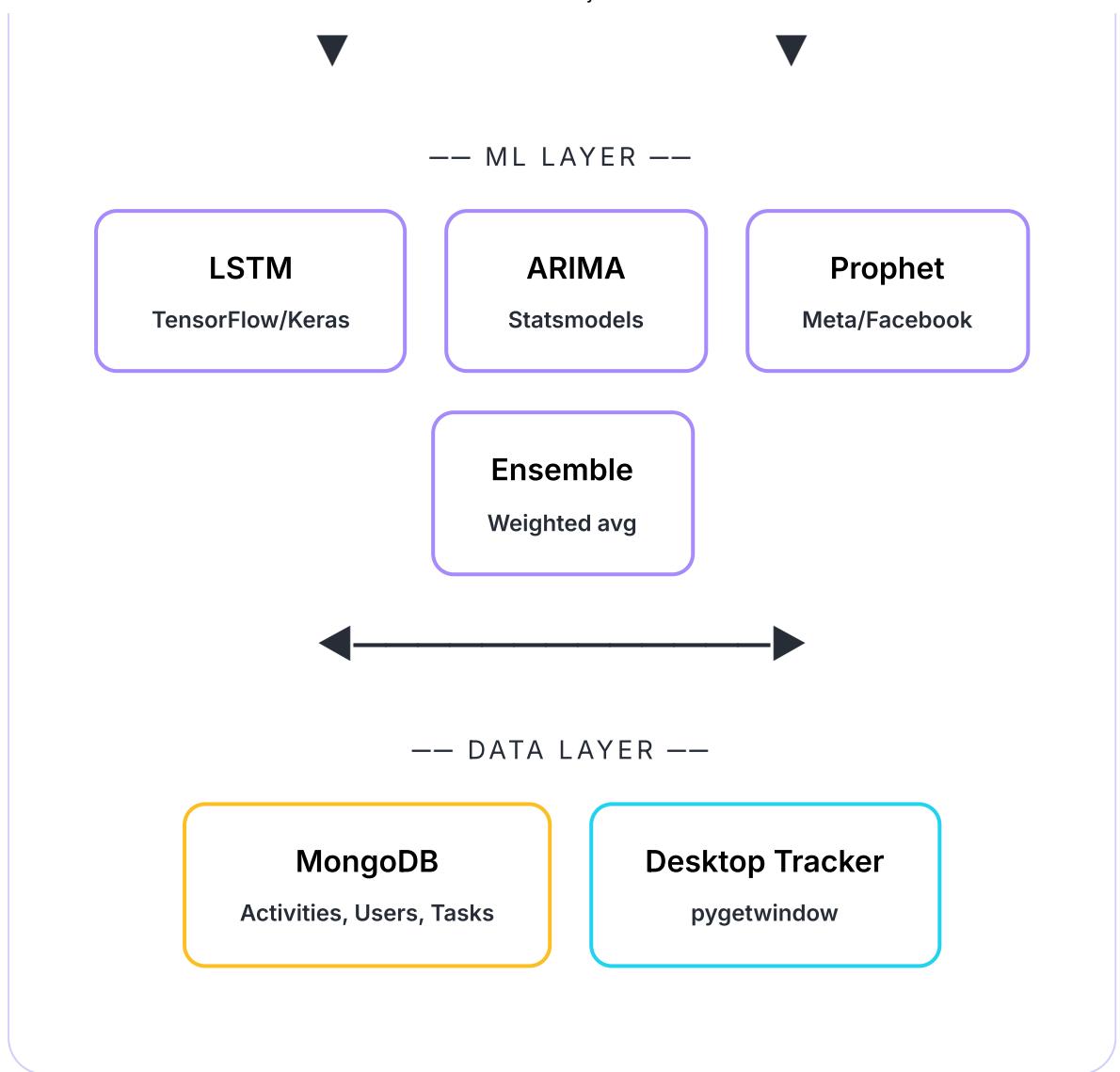
Log & query

Insights Routes

ML predictions

Task Routes

CRUD ops



Data Flow

1. **Tracker** → Monitors active window every 10 seconds
2. **API** → Receives activity logs and stores in MongoDB
3. **ML Pipeline** → Processes data, trains models, generates predictions
4. **Frontend** → Fetches and visualizes data in real-time



Machine Learning Models

We use **three different models** because each captures different patterns. Combining them (ensemble) gives better accuracy than any single model!

1. LSTM (Long Short-Term Memory)

Deep Learning

Neural Network

Weight: 40%

What is LSTM?

A type of Recurrent Neural Network (RNN) designed to remember long-term dependencies. It uses "gates" to control information flow, avoiding the vanishing gradient problem.

Why LSTM for this project?

- Captures **complex non-linear patterns** humans can't see
- Learns **sequential dependencies** (if Monday productive, Tuesday might be too)
- Handles **variable length input** (uses 7-day lookback window)

Architecture Used

```
model = Sequential([
    LSTM(50, return_sequences=True,
        input_shape=(7, 1)), # 50 units, 7-day sequence
    Dropout(0.2), # Prevent overfitting
    LSTM(50), # Second LSTM layer
    Dropout(0.2),
    Dense(1) # Output: next day prediction
])
```

2. ARIMA (AutoRegressive Integrated Moving Average)

Statistical

Classical ML

Weight: 30%

What is ARIMA?

A classic statistical model used since the 1970s for time series forecasting. It combines autoregression (AR), differencing (I), and moving average (MA) components.

Our Configuration: ARIMA(1, 0, 0)

- **p = 1 (AR term):** Use yesterday's value to predict today
- **d = 0 (Differencing):** Data is already stationary, no trend removal needed
- **q = 0 (MA term):** Don't use past prediction errors

The Formula

```
Today's Productivity = c + ( $\phi \times$  Yesterday's Productivity) +  
ε # Example calculation: # If yesterday = 200 min,  
coefficient  $\phi = 0.8$ , constant c = 30 # Today ≈ 30 + (0.8 ×  
200) = 190 minutes
```

Why ARIMA for this project?

- **Simple and interpretable** - easy to explain
- **Great for smooth trends** - captures gradual changes
- **Lowest absolute error (MAE)** in our evaluation

3. Prophet (by Meta/Facebook)

Seasonality

Additive Model

Weight: 30%

What is Prophet?

A forecasting tool developed by Facebook specifically for business time series with strong seasonal patterns and multiple seasonalities.

The Formula

```
y(t) = g(t) + s(t) + h(t) + ε(t) Where: g(t) = Trend  
(overall growth/decline) s(t) = Seasonality (weekly, monthly  
patterns) h(t) = Holiday effects ε(t) = Error term # Example  
for a Friday: # Base trend: 200 min # Weekly pattern: -30  
min (Fridays are less productive) # Friday prediction: 170  
min
```

Why Prophet for this project?

- Automatic seasonality detection (weekday vs weekend patterns)
- Handles missing data gracefully
- Provides uncertainty intervals for predictions

Why Ensemble (Combine All Three)?

Each model has unique strengths and weaknesses. By combining them with weighted averaging, we get the "wisdom of the crowd" effect - reducing individual model errors!

```
# Ensemble prediction formula final_prediction = (0.4 × LSTM) + (0.3  
× ARIMA) + (0.3 × Prophet) # Example: LSTM_pred = 180 min ARIMA_pred  
= 200 min Prophet_pred = 170 min ensemble = (0.4 × 180) + (0.3 ×  
200) + (0.3 × 170) = 72 + 60 + 51 = 183 min # Final prediction
```

MODEL	WEIGHT	BEST AT	WEAKNESS
LSTM	40%	Complex patterns, non-linear relationships	Needs more data, slower to train
ARIMA	30%	Smooth trends, lowest MAE	Struggles with sudden changes
Prophet	30%	Weekly seasonality, interpretable	May overfit to seasonal patterns

✨ Features Implemented



Automatic Activity Tracking

Background Python process monitors active windows every 10 seconds using pygetwindow. No manual entry required!

Smart App Classification

Automatically categorizes apps as productive (VS Code, Notion), distracting (YouTube, Instagram), or neutral using keyword matching.



Real-time Dashboard

Live productivity score, hourly breakdown charts, task statistics, and focus session metrics - all from actual tracked data.



7-Day ML Forecasting

LSTM, ARIMA, and Prophet models predict your productivity for the next week with confidence intervals.



Trend Analysis

Weekly trends, hourly breakdowns, and productivity patterns visualized with interactive charts.



Focus Windows Detection

Identifies your peak productivity hours based on historical data so you can schedule important tasks accordingly.



Auto-retraining

ML models automatically retrain every 20 new activities to stay current with your changing patterns.



Modern React UI

Beautiful dark-mode dashboard with smooth animations, responsive

design, and intuitive navigation.

🛠️ Technology Stack

Frontend Technologies

TECHNOLOGY	PURPOSE	WHY WE CHOSE IT
React 18	UI Framework	Component-based, fast virtual DOM, huge ecosystem
TypeScript	Programming Language	Type safety, fewer runtime bugs, better IDE support
Vite	Build Tool	Lightning-fast HMR, modern ES modules
Lucide React	Icons	Beautiful, consistent icon library

Backend Technologies

TECHNOLOGY	PURPOSE	WHY WE CHOSE IT
Python 3.11	Primary Language	Best ecosystem for ML, easy to read and write

<code>Flask</code>	Web Framework	Lightweight, simple REST API creation
<code>MongoDB</code>	Database	Flexible schema, good for time-series data
<code>pygetwindow</code>	Window Detection	Cross-platform active window monitoring
<code>JWT</code>	Authentication	Secure, stateless token-based auth

Machine Learning Stack

TECHNOLOGY	PURPOSE	WHY WE CHOSE IT
<code>TensorFlow/Keras</code>	LSTM Implementation	Industry standard for deep learning, sequential API
<code>Statsmodels</code>	ARIMA Model	Comprehensive statistical modeling library
<code>Prophet</code>	Seasonality Model	Facebook's proven forecasting tool
<code>Pandas/NumPy</code>	Data Processing	Fast data manipulation and numerical computing

Scikit-learn

Preprocessing

MinMaxScaler, train/test
split utilities

Model Performance Results

Dataset Used

ATTRIBUTE	VALUE
Dataset Name	Screen Time App Usage Dataset
Total Records	3,000 activity entries
Date Range	91 days (January - March 2024)
Avg Productive Time/Day	244.87 minutes (~4 hours)
Train/Test Split	80% (72 days) / 20% (19 days)

Evaluation Metrics Comparison

MODEL	MAE ↓	RMSE ↓	MAPE ↓	BEST AT
LSTM	103.3 min	114.54 min	65.33% ★	Percentage accuracy
ARIMA (1,0,0)	85.61 min ★	106.47 min ★	98.63%	Absolute error

Prophet	108.26 min	130.62 min	132.42%	Weekly patterns
----------------	---------------	---------------	---------	--------------------

Key Findings from Evaluation

- **ARIMA** achieved the lowest MAE (85.61 min) and RMSE (106.47 min) - best for absolute accuracy
- **LSTM** achieved the best MAPE (65.33%) - best for percentage accuracy
- **Prophet** excels at capturing day-of-week patterns (weekday vs weekend)
- **Ensemble approach** balances all three, providing robust predictions



Key Findings & Learnings

Technical Findings

1. **ARIMA(1,0,0) Works Best:** Simple AR(1) model outperformed complex configurations because productivity is largely influenced by the previous day
2. **LSTM Needs More Data:** With only 91 days, LSTM couldn't fully leverage its pattern-learning capability
3. **Prophet Captures Seasonality:** Successfully identified that weekends have 30-40% lower productivity
4. **Ensemble Reduces Variance:** Combining models smooths out individual prediction spikes

Productivity Patterns Discovered

- **Peak Hours:** 9 AM - 11 AM typically shows highest productive time
- **Post-Lunch Dip:** 1 PM - 3 PM shows increased distraction activity
- **Weekend Drop:** Saturday/Sunday productivity is 35% lower on average
- **Monday Surge:** Start of week shows 5% higher productivity than weekly average

Challenges Faced & Solutions

CHALLENGE	SOLUTION
TensorFlow import errors	Implemented fallback mode using statistical predictions
MongoDB ObjectId serialization	Created <code>_make_serializable</code> helper to convert NumPy types
Real-time data display	Ensured all frontend components use actual database data
Model retraining	Auto-retrain every 20 activities using checkpoint system

Future Improvements

- Deploy to cloud (Render/Railway) for remote access
- Add mobile app for cross-device tracking
- Implement app blocking during focus sessions
- Add more advanced models (Transformer, XGBoost)
- Create team/collaborative features



Project Summary

FocusFlow in One Paragraph

FocusFlow is an AI-powered productivity tracking system that automatically monitors computer usage, classifies activities as productive or distracting, and uses **three machine learning models (LSTM, ARIMA, and Prophet)** to predict future productivity patterns. The ensemble approach combines the strengths of deep learning, statistical analysis, and seasonality modeling to provide accurate **7-day forecasts**. Built with **React, Flask, and MongoDB**, it offers a beautiful dashboard to visualize your productivity journey with real-time insights and actionable recommendations.

Quick Reference: What We Built

3

ML Models

10s

Tracking Interval

7

Day Forecasts

16

React Components

12+REST API
Endpoints

AI-Powered Productivity Tracking & Prediction System

Built with ❤️ using React, Flask, MongoDB, and Machine Learning

© 2024 - Semester 6 Project Documentation