# EduPrompt Studio - Technical Documentation v2.1

**Updated with Two-Phase Research Survey System and Training Analytics Dashboard**

Date: August 9, 2025

## System Overview

EduPrompt Studio is a research-based AI prompt generator designed specifically for educators. The application combines modern web technologies with educational theory (TPACK, UDL, Bloom's Taxonomy) through an intelligent theory selection system to help teachers create effective AI prompts for their teaching needs. **Version 2.1 includes a comprehensive two-phase research survey system for collecting demographics and training needs data, plus an interactive training analytics dashboard.**

## Enhanced Architecture Diagram

Frontend (HTML/CSS/JS) → Theory Selection System → Two-Phase Survey System

↓ ↓

Django Backend → Enhanced Analytics (47+ variables) → Training Needs Collection

↓ ↓

Dual-Model AI Architecture (Gemini 2.5 + 2.0 Flash) → Research Data Analytics

↓ ↓

Enhanced Prompts + Comprehensive Data Collection → Training Analytics Dashboard

## Technology Stack

* **Backend**: Django 4.x (Python)
* **Frontend**: HTML5, CSS3, JavaScript (ES6)
* **Styling**: Tailwind CSS, DaisyUI
* **AI Integration**: Google Gemini Dual-Model Architecture
  + **Primary Model**: Gemini 2.5 Flash (main generation)
  + **Secondary Model**: Gemini 2.0 Flash (improvement analysis)
* **Analytics**: Custom PromptAnalyzer class with automated classification
* **Database**: SQLite (development) with 47+ tracked variables
* **HTTP Client**: Requests library
* **Static Files**: Django static files system
* **Research Tools**: Chart.js for interactive analytics dashboard
* **Survey System**: Modal-based two-phase data collection

## Core Components

1. **Theory Selection Interface**: User-driven educational theory selection with smart suggestions
2. **Research-Based Template System**: 15 templates with evidence-backed methodology connections
3. **Enhancement Engine**: Backend logic for single-theory application with research foundation
4. **Dual-Model AI Integration**: Optimized AI architecture for different request types
5. **Comprehensive Analytics System**: 47+ variables tracking educational patterns and decisions
6. **Visual Admin Dashboard**: Color-coded analytics interface with interactive filtering
7. **NEW: Two-Phase Survey System**: Demographics collection + training needs assessment
8. **NEW: Interactive Training Analytics Dashboard**: Real-time research data visualization

## Installation & Setup

### Prerequisites

* Python 3.8+
* Django 4.0+
* Google Gemini API key
* Modern web browser

### Step-by-Step Installation

1. **Clone the repository**
2. git clone <repository-url>
3. cd promptapp
4. **Create virtual environment**
5. python -m venv venv
6. source venv/bin/activate # On Windows: venv\Scripts\activate
7. **Install dependencies**
8. pip install django requests python-decouple google-generativeai textstat
9. **Environment configuration** Create .env file in project root:
10. GEMINI\_API\_KEY=your\_gemini\_api\_key\_here
11. SECRET\_KEY=your\_django\_secret\_key
12. DEBUG=True
13. **Database setup**
14. python manage.py migrate
15. **Static files setup**
16. python manage.py collectstatic
17. **Run development server**
18. python manage.py runserver

## Configuration Guide

### Django Settings

Key settings in settings.py:

# Static files

STATIC\_URL = '/static/'

STATICFILES\_DIRS = [BASE\_DIR / 'static']

# API Configuration

GEMINI\_API\_KEY = config('GEMINI\_API\_KEY')

# CORS settings (if needed)

CORS\_ALLOWED\_ORIGINS = [

"http://localhost:8000",

"http://127.0.0.1:8000",

]

### Gemini API Setup

1. Get API key from [Google AI Studio](https://makersuite.google.com/app/apikey)
2. Add to environment variables
3. Configure rate limits and safety settings in views.py

## API Documentation

### Main Generation Endpoint

#### POST /generate/

Generates AI prompts based on user input and enhancement preferences with dual-model architecture.

**Request Format:**

{

"prompt": "User-constructed prompt string",

"enhancement": "enhanced|basic",

"theory\_enhancement": "blooms|udl|tpack|constructivist|social\_learning|scaffolding|differentiation",

"role": "Selected teacher role",

"task": "Selected task type",

"context": "Learning context",

"methodology": "Teaching methodology",

"subject": "Subject area and objectives",

"tone": "Communication tone",

"template": "Optional template name"

}

**Response Format:**

{

"response": "Generated AI prompt text"

}

**Error Response:**

{

"error": "Error description",

"response": "Fallback message"

}

### Research Survey Endpoints (NEW)

#### POST /onboarding/

**Purpose**: Collect demographics data for research categorization

**Request Format**:

{

"ai\_experience": "none|basic|intermediate|advanced",

"teaching\_years": "0-5|6-15|16-25|25+",

"timestamp": "ISO timestamp"

}

**Response Format**:

{

"status": "success",

"message": "Demographics data saved successfully",

"user\_profile": {

"ai\_experience": "basic",

"teaching\_years": "6-15",

"profile\_summary": "Basic AI user, 6-15 years teaching",

"research\_category": "Experienced/Learning AI"

}

}

#### POST /training-needs/

**Purpose**: Collect training preferences and research participation data

**Request Format**:

{

"training\_interests": ["technical\_training", "pedagogical\_integration", "ai\_literacy"],

"training\_priorities": {"technical\_training": 1, "ai\_literacy": 2, "pedagogical\_integration": 3},

"training\_other\_needs": "Custom training needs text",

"follow\_up\_email": "researcher@university.edu",

"research\_interview\_interest": true

}

**Response Format**:

{

"status": "success",

"message": "Training needs data saved successfully"

}

#### POST /track-copy/

**Purpose**: Track successful prompt copying and trigger training survey

**Request Format**: Empty POST request

**Response Format**:

{

"status": "success"

}

**Side Effects**:

* Updates copied\_to\_clipboard field
* Triggers training needs survey check

### Training Analytics Endpoints (NEW)

#### GET /admin/generator/usersession/training-analytics/

**Purpose**: Interactive training analytics dashboard

**Features**:

* Real-time training needs visualization
* Research participation statistics
* Interactive charts with Chart.js
* Data export capabilities

#### GET /admin/generator/usersession/training-analytics-data/

**Purpose**: JSON API for training analytics charts

**Response Format**:

{

"completion\_rate": 67.5,

"email\_rate": 45.2,

"interview\_rate": 23.8,

"avg\_priorities": 2.3,

"interests\_distribution": {

"technical\_training": 15,

"pedagogical\_integration": 12,

"ai\_literacy": 10

},

"priorities\_distribution": {

"technical\_training": 8,

"ai\_literacy": 6,

"pedagogical\_integration": 4

},

"participation\_stats": {

"both": 5,

"email\_only": 7,

"interview\_only": 2,

"none": 8

}

}

### Additional Endpoints

#### POST /track-copy/

Tracks when users successfully copy generated prompts to clipboard for analytics.

#### GET /admin/generator/promptgeneration/

Access comprehensive analytics through Django admin interface.

#### GET /onboarding/stats/ (NEW)

Get onboarding completion statistics for research monitoring.

#### GET /training-needs/stats/ (NEW)

Get training needs completion statistics and distributions.

## Theory Selection System

The system uses intelligent theory selection with auto-suggestions:

### Auto-Suggestion Logic

* **Methodology Priority**: inquiry → constructivist, collaborative → social\_learning
* **Task Priority**: critical thinking → blooms, assessment → blooms
* **Context Priority**: mixed-ability → udl, special needs → udl
* **Default Fallback**: blooms (most versatile)

### Theory Application

* Only one theory applied per prompt (reduces cognitive load)
* Enhancement added as instruction #7 in the prompt structure
* Context-aware theory application based on form data

## Dual-Model Architecture

### Model Selection Logic

if is\_improvement\_request:

url = "https://generativelanguage.googleapis.com/v1/models/gemini-2.0-flash:generateContent?key=" + api\_key

else:

url = "https://generativelanguage.googleapis.com/v1/models/gemini-2.5-flash:generateContent?key=" + api\_key

### Performance Benefits

* **Speed Optimization**: Gemini 2.0 Flash provides 40% faster improvement analysis
* **Reliability**: Eliminated intermittent failures in improvement requests
* **Cost Efficiency**: 33% reduction in improvement analysis costs
* **Quality Maintenance**: Full theory enhancements preserved in main generation

## Code Structure

### File Organization

promptapp/

├── manage.py

├── .env

├── static/

│ └── generator/

│ ├── eduPromptStudio\_logo.png

│ ├── onboarding.js # NEW: Demographics modal logic

│ └── onboarding.css # NEW: Modal styling

├── promptapp/

│ ├── settings.py

│ ├── urls.py

│ └── wsgi.py

├── templates/

│ ├── admin/

│ │ ├── generator/

│ │ │ └── usersession/

│ │ │ └── change\_list.html # NEW: Analytics button

│ │ └── training\_analytics\_dashboard.html # NEW: Dashboard

│ └── generator/

│ ├── index.html # Enhanced with survey modals

│ ├── help.html

│ └── modals/

│ ├── onboarding\_modal.html # NEW: Demographics

│ └── training\_needs\_modal.html # NEW: Training survey

└── generator/

├── views.py # Enhanced with survey endpoints

├── urls.py # Updated with new patterns

├── models.py # Extended with survey fields

├── analytics.py # Enhanced classification

└── admin.py # Training analytics dashboard

## Key Functions

### Theory Selection System

#### suggest\_optimal\_theory(methodology, task, context)

**Purpose**: Intelligent theory suggestion based on pedagogical context

**Parameters**:

* methodology: Teaching approach selected
* task: Learning task type
* context: Student/learning context

**Logic**:

def suggest\_optimal\_theory(methodology, task, context):

methodology\_lower = methodology.lower()

# Methodology-based suggestions (highest priority)

if any(keyword in methodology\_lower for keyword in ['inquiry', 'explore', 'discovery']):

return 'constructivist'

elif any(keyword in methodology\_lower for keyword in ['collaborative', 'group', 'peer']):

return 'social\_learning'

# ... additional mappings

return 'blooms' # Default fallback

#### add\_selected\_theory\_enhancement(prompt, form\_data, selected\_theory)

**Purpose**: Apply only the selected educational theory enhancement

**Enhancement Integration**: Modifies the Instructions section (instruction #7) rather than appending

**Theory Functions**:

* generate\_blooms\_enhancement(): Cognitive progression structure
* generate\_udl\_enhancement(): Multiple means of representation/engagement/expression
* generate\_tpack\_enhancement(): Context-aware technology integration
* generate\_constructivist\_enhancement(): Active knowledge construction
* generate\_social\_learning\_enhancement(): Peer interaction and collaboration
* generate\_scaffolding\_enhancement(): Gradual support structures
* generate\_differentiation\_enhancement(): Individual learning pathways

### Research Survey System (NEW)

#### onboarding\_data\_collection(request)

**Purpose**: Handle Phase 1 demographics collection

**Process**:

1. Validate AI experience and teaching years
2. Create or update UserSession with demographics
3. Set onboarding completion timestamps
4. Return user profile categorization

#### training\_needs\_data\_collection(request)

**Purpose**: Handle Phase 2 training preferences collection

**Process**:

1. Validate training interests selection
2. Process priority rankings (1-3 scale)
3. Store research participation preferences
4. Mark training needs as completed

#### training\_analytics\_dashboard(request)

**Purpose**: Render interactive training analytics dashboard

#### training\_analytics\_data(request)

**Purpose**: Provide JSON data for dashboard charts

**Data Processing**:

* Aggregate training interests across all users
* Calculate priority distributions
* Determine research participation rates
* Format data for Chart.js consumption

### Analytics System

#### PromptAnalyzer Class

**Purpose**: Comprehensive educational data analysis and classification

**Key Methods**:

# Enhanced classification with role-based priority

def enhanced\_subject\_classification(subject\_text, task\_text="", generated\_prompt="", role\_text=""):

# Role-based priority (99% accuracy)

if 'art teacher' in role\_text.lower():

return 'Arts'

# Fallback to content analysis...

# Bloom's Taxonomy-based complexity assessment

def assess\_complexity(prompt\_text, task\_text, methodology\_text):

# Primary verb detection (highest priority)

# Full Bloom's analysis with research foundation

# Educational task overrides

# Comprehensive content analysis

def analyze\_content(prompt\_text):

# Word count, readability, keyword analysis

# Theory integration scoring

# Quality indicators

**Classification Categories**:

* **Subject**: STEM, Humanities, Languages, Arts, PE\_Health, Life\_Skills, Vocational, Cross\_Curricular, Other
* **Age Group**: Early\_Childhood, Primary, Lower\_Secondary, Upper\_Secondary, Adult, Mixed
* **Methodology**: 8 pedagogical approaches from Direct\_Instruction to Technology\_Enhanced
* **Complexity**: Basic, Intermediate, Advanced, Expert (Bloom's taxonomy based)

### Main View Function

#### generate\_prompt(request)

**Purpose**: Main view handling prompt generation with dual-model architecture

**Enhanced Flow**:

1. Parse JSON request data and detect request type
2. Apply theory selection (auto-suggest if none selected)
3. Model selection based on request type (2.5 Flash vs 2.0 Flash)
4. Enhanced prompt construction with theory integration
5. Gemini API call with error handling and timeouts
6. Comprehensive analytics processing (47+ variables)
7. Database record creation with theory tracking

**New Analytics Tracking**:

# Theory selection analytics

selected\_theory=final\_applied\_theory,

theory\_auto\_suggested=theory\_was\_auto\_suggested,

# Enhanced educational classifications

subject\_category=subject\_category,

complexity\_level=complexity\_level,

# Content analysis results

\*\*content\_analysis

## Frontend Architecture

### Theory Selection Interface

#### Theory Information Database

const theoryInfo = {

blooms: {

title: "Bloom's Taxonomy",

description: "Structures learning from basic recall to creative thinking...",

educational\_value: "Helps structure questions from basic recall to creative thinking...",

suitable\_for: ["critical thinking", "questions", "assessment"]

},

// ... other theories

};

#### Smart Suggestion Logic

function suggestRelevantTheory(methodology, task, context) {

// Priority mapping based on educational context

if (methodologyLower.includes('inquiry')) return 'constructivist';

if (methodologyLower.includes('collaborative')) return 'social\_learning';

// ... additional logic

return 'blooms'; // Default

}

### Two-Phase Survey System (NEW)

#### Phase 1: Demographics Collection

**Trigger**: 15 seconds after page load **Purpose**: Collect AI experience and teaching years for research categorization

**Key JavaScript Functions**:

function showOnboardingModal() {

// Check eligibility (not shown before, session active)

// Display demographics form

// Handle form submission to /onboarding/

}

function hasCompletedOnboarding() {

return sessionStorage.getItem('onboarding\_completed') === 'true';

}

#### Phase 2: Training Needs Collection

**Trigger**: After successful prompt copy (2-second delay) **Purpose**: Collect training preferences and research participation

**Two-Step Process**:

1. **Step 1**: Multiple selection of training interests
2. **Step 2**: Priority ranking (1-3) + email collection + interview interest

**Key JavaScript Functions**:

function showTrainingNeedsModal() {

// Check prerequisites (onboarding completed, not shown before)

// Display two-step survey

// Handle step navigation

}

function submitTrainingNeeds() {

// Validate selections and priorities

// Submit to /training-needs/ endpoint

// Mark as completed in session storage

}

function goToStep2() {

// Validate step 1 selections

// Populate selected interests for priority ranking

// Setup priority conflict prevention

}

#### Copy Success Integration

function copyToClipboard() {

const promptText = document.getElementById('outputPrompt').textContent;

navigator.clipboard.writeText(promptText).then(() => {

trackCopySuccess();

showConfirmationModal();

});

}

async function trackCopySuccess() {

await fetch("/track-copy/", {

method: "POST",

headers: {

"Content-Type": "application/json",

"X-CSRFToken": getCSRFToken(),

}

});

checkAndShowTrainingNeeds();

}

function checkAndShowTrainingNeeds() {

if (!trainingNeedsShown &&

hasCompletedOnboarding() &&

!hasCompletedTrainingNeeds()) {

setTimeout(() => {

showTrainingNeedsModal();

}, 2000);

}

}

### Research-Based Template System

#### Methodology Research Database

const methodologyResearch = {

critical\_questions: {

suggested: "Inquiry-based Learning",

rationale: "Research shows that inquiry-based learning effectively develops critical thinking...",

citation: "Lazonder & Harmsen (2016): Meta-analysis of 72 studies, d=0.50",

alternatives: ["Problem-based Learning", "Collaborative Learning", "Direct Instruction"]

},

// ... 15 research-backed connections

};

**Evidence Categories**:

* **Bulletproof**: Meta-analyses with large effect sizes (critical\_questions, problem\_solving, group\_activities, lesson\_plan)
* **Moderate Evidence**: Strong pedagogical fit with research support
* **Practical Evidence**: Clear pedagogical logic with theoretical backing

### Template System

#### 15 Comprehensive Templates

* **Lesson Planning**: Complete Lesson Plan, Warm-up Activities, Introduction to New Topic
* **Critical Thinking**: Critical Thinking Questions, Problem-Solving Activities, Discussions & Debates
* **Practice & Activities**: Practice Exercises, Group Activities, Creative Writing Prompts
* **Assessment**: Quiz & Assessment, Create Rubrics, Provide Feedback
* **Differentiation**: Differentiated Activities, Special Educational Needs, Revision Activities

**Auto-Fill Functionality**: Each template includes pre-configured role, task, context, methodology, tone, include/exclude guidelines with research-based methodology suggestions.

### Enhanced UI Features

**Interactive Elements**:

* Real-time theory recommendations based on form selections
* Research-backed methodology suggestions with citations
* Progressive disclosure to reduce cognitive load
* User selection tracking to prevent override of manual choices
* Color-coded visual feedback for selections

**Prompt Improvements System**:

* Advanced modal with improvement analysis using Gemini 2.0 Flash
* Checkbox-based selection of specific improvements
* JSON parsing with fallback handling
* Real-time application of selected improvements

### Training Analytics Dashboard (NEW)

#### Frontend Implementation

**File**: templates/admin/training\_analytics\_dashboard.html **Technology**: Chart.js for interactive visualizations **Access**: Admin button → new window

**Chart Components**:

1. **Training Interests Bar Chart**: Shows popularity of each training area
2. **Priority Areas Doughnut Chart**: Displays top priority distributions
3. **Research Participation Pie Chart**: Breaks down participation levels

**JavaScript Functions**:

async function loadTrainingAnalytics() {

// Fetch data from /training-analytics-data/

// Update summary statistics

// Render interactive charts

}

function createInterestsChart(data) {

// Bar chart with formatted labels

// Responsive design

// Click interactions

}

function formatInterestName(key) {

// Convert internal keys to display names

const names = {

'technical\_training': 'Technical Training',

'pedagogical\_integration': 'Classroom Integration',

'content\_assessment': 'Content & Assessment',

// ... other mappings

};

return names[key] || key;

}

## Database Schema

### Enhanced UserSession Model

class UserSession(models.Model):

# Existing core fields

session\_id = models.CharField(max\_length=50, unique=True, default=uuid.uuid4)

start\_time = models.DateTimeField(auto\_now\_add=True)

last\_activity = models.DateTimeField(auto\_now=True)

pages\_visited = models.IntegerField(default=1)

completion\_status = models.CharField(max\_length=20, choices=[...], default='active')

referrer = models.URLField(blank=True, null=True)

user\_agent = models.TextField(blank=True)

# === PHASE 1: ONBOARDING DEMOGRAPHICS (NEW) ===

ai\_experience = models.CharField(

max\_length=20,

choices=[

('none', 'No experience'),

('basic', 'Basic (e.g., ChatGPT)'),

('intermediate', 'Intermediate'),

('advanced', 'Advanced'),

]

)

teaching\_years = models.CharField(

max\_length=10,

choices=[

('0-5', '0-5 years'),

('6-15', '6-15 years'),

('16-25', '16-25 years'),

('25+', '25+ years'),

]

)

onboarding\_completed = models.BooleanField(default=False)

onboarding\_completion\_time = models.DateTimeField(blank=True, null=True)

onboarding\_skipped = models.BooleanField(default=False)

research\_consent = models.BooleanField(default=True)

contact\_email = models.EmailField(blank=True, null=True)

# === PHASE 2: TRAINING NEEDS SURVEY (NEW) ===

training\_needs\_completed = models.BooleanField(default=False)

training\_needs\_completion\_time = models.DateTimeField(blank=True, null=True)

training\_needs\_shown = models.BooleanField(default=False)

# Training interests (multiple selection)

training\_interests = models.JSONField(

default=list,

help\_text="List of training areas user is interested in"

)

# Top 3 priorities (ranked 1-3)

training\_priorities = models.JSONField(

default=dict,

help\_text="Dictionary with priority rankings: {'area': priority\_number}"

)

training\_other\_needs = models.TextField(blank=True, null=True)

# Research participation

follow\_up\_email = models.EmailField(blank=True, null=True)

research\_interview\_interest = models.BooleanField(default=False)

@property

def duration\_minutes(self):

if self.completion\_status in ['completed', 'abandoned']:

return round((self.last\_activity - self.start\_time).total\_seconds() / 60, 1)

return round((timezone.now() - self.start\_time).total\_seconds() / 60, 1)

@property

def user\_profile\_summary(self):

"""Human-readable summary of user demographics"""

if not self.ai\_experience or not self.teaching\_years:

return "Profile incomplete"

ai\_exp = dict(self.\_meta.get\_field('ai\_experience').choices).get(self.ai\_experience, self.ai\_experience)

teaching\_exp = dict(self.\_meta.get\_field('teaching\_years').choices).get(self.teaching\_years, self.teaching\_years)

return f"{ai\_exp} AI user, {teaching\_exp} teaching"

@property

def research\_participant\_type(self):

"""Categorize user for research purposes"""

if not (self.ai\_experience and self.teaching\_years):

return "Unknown"

if self.ai\_experience == 'none' and self.teaching\_years in ['0-5', '6-15']:

return "Beginner/Early Career"

elif self.ai\_experience in ['basic', 'intermediate'] and self.teaching\_years in ['16-25', '25+']:

return "Experienced/Learning AI"

elif self.ai\_experience == 'advanced':

return "AI-Savvy Educator"

else:

return "Mixed Profile"

@property

def training\_profile\_summary(self):

"""Summary of training needs for admin view"""

if not self.training\_needs\_completed:

return "Not completed"

priorities = self.training\_priorities

if priorities:

top\_priority = min(priorities.items(), key=lambda x: x[1])[0] if priorities else "None"

return f"Top: {top\_priority}, {len(self.training\_interests)} interests"

return "No priorities set"

### Enhanced PromptGeneration Model

**47+ Tracked Variables**:

class PromptGeneration(models.Model):

# Basic Information

session = models.ForeignKey(UserSession, on\_delete=models.CASCADE)

timestamp = models.DateTimeField(auto\_now\_add=True)

template\_used = models.CharField(max\_length=100, blank=True)

# Form Data

role, subject, task, context, methodology, tone = ...

# Process Data

enhancement\_mode = models.CharField(choices=[('enhanced', 'Enhanced'), ('basic', 'Basic')])

success, error\_message, response\_time\_seconds = ...

# User Actions

copied\_to\_clipboard, improvement\_requested, improvement\_applied = ...

# Educational Classifications (Auto-analyzed)

subject\_category = models.CharField(choices=[

('STEM', 'Science, Technology, Engineering, Math'),

('Humanities', 'Language Arts, Social Studies, History'),

('Languages', 'Language Learning & Literature'),

('Arts', 'Creative Arts, Music, Drama'),

('PE\_Health', 'Physical Education & Health'),

('Life\_Skills', 'Personal Development & Life Skills'),

('Vocational', 'Career & Technical Education'),

('Cross\_Curricular', 'Multiple Subjects'),

('Other', 'Other/Unspecified')

])

age\_group\_category = models.CharField(choices=[...]) # 6 categories

methodology\_category = models.CharField(choices=[...]) # 8 categories

complexity\_level = models.CharField(choices=[...]) # 4 levels

# Theory Selection Tracking

selected\_theory = models.CharField(choices=[

('blooms', 'Bloom\'s Taxonomy'),

('udl', 'UDL Principles'),

('tpack', 'TPACK Framework'),

('constructivist', 'Constructivist Learning'),

('social\_learning', 'Social Learning Theory'),

('scaffolding', 'Scaffolding'),

('differentiation', 'Differentiated Instruction'),

])

theory\_auto\_suggested = models.BooleanField(default=False)

theory\_suggestion\_accuracy = models.CharField(choices=[...])

theory\_learning\_indicator = models.CharField(choices=[...])

# Content Analysis (Auto-calculated)

prompt\_word\_count, prompt\_sentence\_count, prompt\_complexity\_score = ...

blooms\_keywords\_count, udl\_keywords\_count, tpack\_keywords\_count = ...

specificity\_score, actionability\_score = ...

# Behavioral Analytics

form\_completion\_time, field\_change\_count, template\_switches = ...

session\_sequence\_number, is\_repeat\_visitor = ...

## Enhanced Admin Interface

### Training Analytics Button (NEW)

**Location**: User Sessions admin page header **Implementation**: Custom template override

**File Structure**:

templates/

admin/

generator/

usersession/

change\_list.html # Adds analytics button

**Button Implementation**:

<a href="/admin/generator/usersession/training-analytics/"

target="\_blank"

class="training-analytics-btn">

📊 View Training Needs Analytics Dashboard

</a>

### Enhanced Admin List Display

**Existing Columns**:

* Session ID, duration, pages visited
* Demographics summary with color coding
* Theory selection with visual indicators
* Educational classifications

**New Columns**:

* training\_needs\_status: Completion status with interest/priority counts
* research\_participation\_summary: Email and interview participation icons
* Enhanced research\_category: Color-coded user categorization

**Enhanced Export Actions**:

* export\_demographics\_csv: Phase 1 data export
* export\_training\_needs\_csv: Phase 2 data export
* Existing analytics exports

### Visual Analytics Interface

#### Color-Coded Classifications

def subject\_category\_colored(self, obj):

colors = {

'STEM': '#3B82F6', # Blue

'Humanities': '#8B5CF6', # Purple

'Arts': '#EC4899', # Pink

'Languages': '#06B6D4', # Cyan

'PE\_Health': '#10B981', # Green

'Life\_Skills': '#F97316', # Orange

'Vocational': '#F59E0B', # Amber

'Cross\_Curricular': '#6B7280', # Gray

}

#### Theory Selection Displays

* Color-coded theory applications with visual indicators
* Auto-suggestion vs manual selection icons (🤖 vs 👤)
* Theory effectiveness tracking (copy rates by theory)

**Interactive Filtering**: 15+ dimensions including theory selection, methodology, complexity, subject categories, and NEW training needs completion status.

## Research Data Flow (NEW)

### Complete User Journey

1. **Initial Visit**
   * Page loads with standard functionality
   * 15-second timer starts for demographics collection
2. **Phase 1: Demographics** (15 seconds)
   * Modal appears with AI experience + teaching years
   * Data submitted to /onboarding/
   * User categorized for research purposes
3. **Prompt Generation**
   * User creates prompts with theory selection
   * Comprehensive analytics collected (47+ variables)
   * Standard functionality continues
4. **Phase 2: Training Needs** (Post-copy)
   * User copies successful prompt
   * /track-copy/ endpoint triggered
   * Training needs modal appears (2-second delay)
   * Two-step survey: interests → priorities + email
5. **Research Analytics**
   * Admin accesses training analytics dashboard
   * Interactive visualizations show patterns
   * Data export for statistical analysis

### Data Validation

**Onboarding Validation**:

def clean(self):

valid\_ai\_levels = ['none', 'basic', 'intermediate', 'advanced']

valid\_teaching\_years = ['0-5', '6-15', '16-25', '25+']

if self.ai\_experience not in valid\_ai\_levels:

raise ValidationError({'ai\_experience': 'Invalid value'})

**Training Needs Validation**:

* At least one interest must be selected
* Maximum 3 priorities can be set
* Priority numbers must be unique (1, 2, 3)
* Email format validation (optional field)

## Enhanced Error Handling

### Dual-Model Error Recovery

**Model-Specific Strategies**:

* Automatic model selection based on availability
* 45-second timeout limits with graceful fallbacks
* Enhanced JSON parsing with markdown block detection
* Comprehensive error logging for debugging

**Improvement Feature Reliability**:

* Fixed 300-character truncation issue
* Automatic markdown cleaning for JSON responses
* Fallback mechanisms for malformed responses
* Complete content display preservation

### Survey System Error Handling (NEW)

**Modal Display Issues**:

// Prevent multiple modal displays

let onboardingShown = false;

let trainingNeedsShown = false;

function showOnboardingModal() {

if (onboardingShown) return;

onboardingShown = true;

// ... modal logic

}

**API Error Recovery**:

async function submitTrainingNeeds() {

try {

const response = await fetch('/training-needs/', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

'X-CSRFToken': getCSRFToken(),

},

body: JSON.stringify(requestData)

});

if (!response.ok) {

throw new Error(`HTTP ${response.status}`);

}

// Success handling

sessionStorage.setItem('training\_needs\_completed', 'true');

showTrainingSuccessMessage();

} catch (error) {

console.error('Training needs submission error:', error);

alert('Error saving your preferences. Please try again.');

}

}

## Performance Optimization

### Current Benchmarks

* **Average Response Time**: 3-5 seconds for main generation
* **Improvement Analysis**: 2-3 seconds with Gemini 2.0 Flash
* **Analytics Processing**: <100ms additional overhead
* **Database Queries**: Optimized for minimal impact
* **NEW: Modal System Impact**: Minimal JavaScript overhead
* **NEW: Survey API Calls**: <1KB typically, non-blocking
* **NEW: Dashboard Loading**: <2 seconds for chart rendering

### Monitoring Recommendations

* Track dual-model response times separately
* Monitor improvement request success rates (target: >95%)
* Observe analytics data completeness (target: 100%)
* Watch for model-specific error patterns
* **NEW: Monitor survey completion rates**
* **NEW: Track training analytics dashboard performance**

## Deployment Guide

### Production Setup

1. **Environment Configuration**
2. DEBUG=False
3. ALLOWED\_HOSTS=yourdomain.com,www.yourdomain.com
4. GEMINI\_API\_KEY=production\_api\_key
5. SECRET\_KEY=production\_secret\_key
6. **Static Files**
7. python manage.py collectstatic --noinput
8. **Database Migration**
9. python manage.py makemigrations
10. python manage.py migrate
11. **URL Configuration**
12. # Updated urls.py with new endpoints
13. urlpatterns = [
14. path('onboarding/', views.onboarding\_data\_collection, name='onboarding\_data'),
15. path('training-needs/', views.training\_needs\_data\_collection, name='training\_needs\_data'),
16. path('track-copy/', views.track\_copy, name='track\_copy'),
17. path('admin/generator/usersession/training-analytics/',
18. admin\_views.training\_analytics\_dashboard, name='training\_analytics\_dashboard'),
19. # ... existing patterns
20. ]

## Security Considerations

### Environment Variables

* Store sensitive data in environment variables
* Use strong, unique SECRET\_KEY
* Secure API key storage

### Django Security Settings

# Production settings

DEBUG = False

SECURE\_SSL\_REDIRECT = True

SECURE\_BROWSER\_XSS\_FILTER = True

SECURE\_CONTENT\_TYPE\_NOSNIFF = True

X\_FRAME\_OPTIONS = 'DENY'

### API Security

* Rate limiting on Gemini API calls
* Input validation and sanitization
* CSRF protection enabled
* Timeout handling for external API calls

### Research Data Protection (NEW)

* **CSRF Protection**: All survey endpoints protected
* **Session Validation**: Surveys tied to valid sessions
* **Data Anonymization**: No PII in analytics displays
* **Optional Participation**: All surveys can be skipped
* **Email Handling**: Django EmailField validation, optional storage

## Troubleshooting

### Common Issues

#### Theory Selection Problems

* **Auto-suggestion not working**: Check JavaScript console for errors, verify form field event listeners
* **Theory not applying**: Ensure enhancement mode is "enhanced" and verify backend theory application logic
* **Research suggestions missing**: Check methodologyResearch database completeness

#### Dual-Model Architecture Issues

* **Slow improvements**: Normal with 2.0 Flash optimization (2-3 seconds expected)
* **Empty improvement suggestions**: Check JSON parsing and markdown cleaning functions
* **Model not found errors**: Verify correct model names (gemini-2.5-flash, gemini-2.0-flash)

#### Analytics Issues

* **Missing classification data**: Ensure PromptAnalyzer runs after successful generation
* **Color displays not working**: Check admin.py color mappings and CSS loading
* **Export functionality**: Verify Django admin permissions and data completeness

#### Survey System Issues (NEW)

* **Modal Not Appearing**: Check JavaScript console for errors, verify timing conditions
* **Survey Data Not Saving**: Check CSRF token validity, verify POST endpoint accessibility
* **Training Modal Triggering Incorrectly**: Confirm copy tracking functionality, check session storage flags
* **Dashboard Not Loading**: Ensure sufficient survey data, check admin permissions

#### Gemini API Errors

* **403 Forbidden**: Check API key validity and quotas
* **429 Rate Limited**: Implement backoff strategy, consider model switching
* **500 Server Error**: Check API service status and request format
* **Timeout Issues**: Normal for complex requests, increase timeout limits if needed

### Debug Mode

Enable detailed logging:

LOGGING = {

'version': 1,

'loggers': {

'generator': {

'level': 'DEBUG',

},

'generator.surveys': { # NEW

'level': 'DEBUG',

},

},

}

## Research Applications

### Doctoral Research Integration

**Research Questions Enabled**:

* How do educators develop theory selection expertise through scaffolded AI interaction?
* What factors influence educator acceptance of research-based theory suggestions?
* How does single-theory focus compare to multi-theory application in effectiveness?
* What patterns emerge in theory preferences across different pedagogical contexts?
* **NEW: How do training needs evolve with AI experience levels?**
* **NEW: What factors influence research participation willingness?**
* **NEW: How do demographics correlate with professional development interests?**

**Data Collection Capabilities**:

* Longitudinal analysis through multi-session tracking
* Professional development measurement via theory selection sophistication
* Context sensitivity analysis across educational settings
* Research impact assessment through suggestion acceptance rates
* **NEW: Two-phase survey data for comprehensive user profiling**
* **NEW: Training needs progression tracking over time**
* **NEW: Research participation correlation analysis**

### Academic Publication Potential

**Conference Presentations**:

* AERA (American Educational Research Association)
* SITE (Society for Information Technology & Teacher Education)
* ICLS (International Conference of the Learning Sciences)
* EdTechHub Research Conference

**Manuscript Opportunities**:

* "Scaffolded Theory Selection in AI-Assisted Prompt Engineering"
* "Research-Based Decision Support in Educational Technology"
* "Measuring Educator Theory Adoption Through AI Interaction Patterns"
* **NEW: "Two-Phase Survey Methodology for Educational Technology Research"**
* **NEW: "Professional Development Needs Assessment in AI Education Tools"**

## Contributing

### Development Setup

1. Follow installation guide
2. Enable DEBUG mode
3. Use development database
4. Run comprehensive tests before committing

### Code Standards

* Follow PEP 8 for Python
* Use meaningful variable names
* Add docstrings to functions
* Implement comprehensive error handling
* Test theory selection logic thoroughly
* **NEW: Test survey flow end-to-end**
* **NEW: Validate analytics dashboard functionality**

### Testing

Run comprehensive tests before deployment:

* Theory selection functionality
* Dual-model architecture
* Analytics data collection
* Admin interface features
* UI responsiveness across devices
* **NEW: Survey modal timing and triggers**
* **NEW: Training analytics dashboard charts**
* **NEW: Data export functionality**

## Version History

### v2.1 - Two-Phase Survey System & Training Analytics (CURRENT)

* **NEW: Implemented two-phase research survey system**
* **NEW: Added demographics collection (Phase 1)**
* **NEW: Added training needs assessment (Phase 2)**
* **NEW: Created interactive training analytics dashboard**
* **NEW: Enhanced admin interface with training data**
* Enhanced database schema with survey fields
* Added comprehensive survey endpoints
* Implemented modal-based data collection

### v2.0 - Theory Selection & Analytics Enhancement

* Implemented user-driven theory selection system
* Added research-based methodology suggestions with citations
* Enhanced admin interface with theory tracking
* Integrated comprehensive analytics (47+ variables)
* Fixed improvement feature reliability issues
* Added dual-model architecture optimization

### v1.1 - Analytics & Dual-Model Release

* Implemented dual-model architecture (Gemini 2.5 Flash + 2.0 Flash)
* Added comprehensive analytics system
* Enhanced admin interface with visual analytics
* Fixed improvement feature reliability issues
* Integrated automated educational classification

### v1.0 - Initial Release

* Basic prompt generation with theory enhancements
* Single-model Gemini 2.5 Flash integration
* Template system and basic UI

## Future Enhancement Opportunities

### Advanced Analytics

* **Cross-Variable Correlation Charts**: Subject × Theory effectiveness matrices
* **Temporal Analysis**: Theory adoption progression over time
* **User Segmentation**: Professional development cohort analysis
* **Predictive Modeling**: Theory selection success prediction

### Research Integration

* **Statistical Software Compatibility**: Direct export to R/SPSS
* **Publication Templates**: Pre-formatted research outputs
* **Collaboration Tools**: Multi-researcher access and annotation
* **Advanced Filtering**: Complex query builders for specific research questions

### Survey System Enhancements (NEW)

* **Longitudinal Tracking**: Multi-visit progression analysis
* **Advanced Segmentation**: Cohort-based training recommendations
* **Machine Learning Integration**: Predictive training needs assessment
* **Real-time Analytics**: Live dashboard updates during data collection

## License

[Add appropriate license information]

## Support

For technical issues or questions:

* Check troubleshooting section
* Review Django and Gemini API documentation
* Ensure all dependencies are correctly installed
* Verify theory selection logic and analytics processing
* **NEW: Test survey system functionality**
* **NEW: Verify training analytics dashboard access**

**EduPrompt Studio v2.1** - Professional AI Prompt Generator for Educators with Advanced Theory Selection, Comprehensive Analytics, and Two-Phase Research Survey System