**Midterm Assignment: Domain-Specific AI Assistant Project Plan**

Overview

In this group assignment, you will collaborate with your established team members to develop a comprehensive project plan for creating a domain-specific AI assistant. This project plan will serve as the foundation for your group's final project implementation. Together, you'll apply concepts from data preprocessing, feature engineering, and AI system design to create a practical and valuable assistant for a specific domain.

Learning Objectives

* Apply data science concepts to real-world AI applications
* Design appropriate data preprocessing workflows for different types of AI data
* Develop feature engineering strategies for specialized domains
* Create a comprehensive project plan that addresses both technical and practical considerations
* Collaborate effectively within a team environment to develop an AI solution

Assignment Details

Deliverable Format

* A single PDF document named according to the course convention: MD\_GroupName\_ITAI2377.pdf
* Professional formatting with clear section headers, diagrams, and citations where appropriate
* 8-12 pages in length (excluding references and appendices)

Project Plan Components

Your group's project plan must include the following sections:

1. Executive Summary (5%)

* Brief overview of your assistant's purpose and value proposition
* Summary of key functionalities and target users
* High-level approach to implementation

2. Project Definition (20%)

* **Domain Selection**: Identify and justify a specific domain for your assistant (healthcare, education, customer service, etc.)
* **Problem Statement**: Clearly articulate the problem your assistant will solve
* **Core Functionalities**: Define 3-5 specific capabilities your assistant will provide
* **Target Users**: Describe who will use your assistant and what their needs are
* **Value Proposition**: Explain how your assistant will provide value in this domain

3. Data Requirements Analysis (25%)

* **Data Types**: Identify what types of data are needed (structured, unstructured, text, images)
* **Data Sources**: Specify where you will obtain the necessary data (public datasets, APIs, synthetic data)
* **Data Volume and Velocity**: Estimate how much data you will need and how often it will be updated
* **Data Quality Requirements**: Define what makes data "good enough" for your application
* **Potential Data Challenges**: Identify issues you might encounter with data acquisition or quality
* **Data Schema**: Create a visual representation of your data structure

4. Processing Pipeline Design (25%)

* **Data Preprocessing Workflow**: Design a step-by-step process for cleaning and preparing your data
* **Feature Engineering Approach**: Describe specific features you will create and why they matter for your domain
* **Processing Pipeline Diagram**: Create a flowchart showing the complete data preparation process
* **Data Transformation Techniques**: Specify methods for handling different data types (e.g., text normalization, image preprocessing)
* **Infrastructure Considerations**: Address how your pipeline will handle the required data volume

5. Implementation Strategy (15%)

* **Technical Stack**: Specify the tools, libraries, and technologies you will use
* **Development Timeline**: Create a schedule with key milestones for the final implementation
* **Team Member Responsibilities**: Clearly define who will handle which aspects of the implementation
* **Resource Requirements**: Identify computational and human resources needed
* **Implementation Challenges**: Anticipate technical hurdles and propose solutions
* **Integration Approach**: Explain how components will work together in the final system

6. Evaluation Framework (10%)

* **Success Metrics**: Define how you will measure the effectiveness of your assistant
* **Testing Strategy**: Describe how you will validate each component
* **User Feedback Mechanism**: Plan how you will incorporate user input
* **Performance Benchmarks**: Establish minimum acceptable performance levels
* **Continuous Improvement Process**: Outline how you will refine your assistant over time

Domain Options

Your group may choose any specialized domain for your assistant, but here are some suggestions:

1. **Healthcare Assistant**: Help patients understand medical information, track symptoms, or manage medications
2. **Educational Tutor**: Assist students in learning specific subjects, generating practice problems, or explaining concepts
3. **Financial Advisor**: Help users with budgeting, investment basics, or understanding financial terms
4. **Legal Information Assistant**: Provide basic information about laws, legal procedures, or document preparation
5. **Environmental Monitoring**: Analyze environmental data and provide insights or recommendations
6. **Small Business Helper**: Assist with marketing, customer service, or basic business operations
7. **Personal Wellness Coach**: Support users in fitness, nutrition, or mental wellness goals
8. **Technical Support Agent**: Help users troubleshoot common problems with devices or software

Technical Approaches

Your group should select one of these approaches for your assistant’s implementation:

1. **Retrieval-Augmented System**: Using vector embeddings to find relevant information from a knowledge base
2. **Few-Shot Learning**: Creating effective examples to guide AI responses
3. **Simple Rule-Based System**: For domains with clear guidelines and protocols

Your plan should explain your choice and how it aligns with your domain and data requirements.

Evaluation Criteria

Your group's project plan will be evaluated based on:

* **Thoroughness**: Comprehensive coverage of all required sections
* **Feasibility**: Realistic scope given the constraints of time and resources
* **Technical Soundness**: Appropriate application of data science concepts
* **Clarity**: Well-organized presentation with clear explanations
* **Creativity**: Innovative approach to solving a meaningful problem
* **Domain Understanding**: Demonstrated knowledge of the selected field
* **Collaboration Quality**: Evidence of equitable contribution from all team members

Submission Instructions

1. Submit your completed group PDF via Canvas by **July 5, 2025, 11:59 PM**.
2. Include a clear list of all team members in your document.
3. Be prepared to use this plan as the foundation for your group's final project implementation.

Group Work Guidelines

* Each group member should have clear, defined responsibilities in the project plan
* Include a brief statement of contribution outlining each member's role in developing the plan
* All team members should review and approve the final submission
* Use collaborative tools (Google Docs, GitHub, etc.) to facilitate group work
* If issues arise within your group, contact the instructor promptly for assistance

Resources

* Course materials on data preprocessing, feature engineering, and model evaluation
* Public dataset repositories: Kaggle, UCI Machine Learning Repository, Hugging Face Datasets
* Google Colab (for prototyping your data pipeline components)
* Documentation for relevant Python libraries (NLTK, spaCy, Pandas, TensorFlow, etc.)

Important Notes

* Focus on the data science aspects of your assistant rather than complex AI architectures
* Your plan should be realistic for implementation with the limited resources available (especially free Colab)
* Consider the ethical implications of your assistant, particularly for sensitive domains
* This plan will directly inform your final project, so make choices your group can reasonably implement

Start of Group Document -------

🧩 Data Types

1. **Text Data**
   * User queries and questions (e.g., “Why won’t my thermostat connect?”)
   * Troubleshooting documentation (from device manuals, support pages)
   * Automation scripts (YAML or JSON)
   * Error messages and logs from smart home hubs/devices
2. **Structured Data**
   * Device metadata (brand, model, firmware version, connection type)
   * User preferences or settings (opt-in/opt-out for privacy features)
3. **Semi-Structured Data**
   * Config files (e.g., .yaml for Home Assistant)
   * Privacy policies (often HTML, PDF, or scraped markdown)
4. **Optional: Visual or Image Data**

* Screenshots of smart home dashboards or error pop-ups (for OCR, if your team explores that route)

🌐 Data Sources

1. **Public Documentation & FAQs**
   * Smart home brand websites: Google Nest, Philips Hue, Amazon Alexa, etc.
   * Community wikis or forums (e.g., Home Assistant, Reddit threads)
2. **Open Repositories**
   * GitHub repos for smart home automations (search: “home-assistant automation yaml”)
   * Sample logs, config files, and test scenarios from open-source smart home environments
3. **Synthetic Data (Generated by Your Team)**
   * Simulated device logs for common failure scenarios
   * Variations of user queries phrased differently
   * Sample privacy policies or automation rules
4. **Optional APIs or Tools**

* Home Assistant API (to simulate device state)
* OpenAI/nomic embedding tools (for testing retrieval if connected to docs)

📊 **Data Requirements Analysis – SmartHome Assist**

1. **Data Types**

| **Category** | **Description** |
| --- | --- |
| **Text Data** | User queries (natural language), device manuals, troubleshooting docs, error logs |
| **Structured Data** | Device metadata (brand, model, firmware version), user settings/preferences |
| **Semi-Structured** | YAML/JSON smart home configs, privacy policy excerpts, usage templates |
| **Optional Visual** | Screenshots of app dashboards or error messages for optional OCR and visual debugging |

2. **Data Sources**

| **Source Type** | **Examples** |
| --- | --- |
| **Public Documentation** | Smart home FAQs and manuals (Google Home, Alexa, Philips Hue, etc.) |
| **Open Repositories** | GitHub datasets with automation examples and sample configurations (Home Assistant projects) |
| **User Simulations** | Manually crafted logs, scripts, and queries for controlled testing scenarios |
| **Privacy Policy Extracts** | Policies scraped from official smart device pages or collected via Common Crawl subsets |

3. **Data Volume and Velocity**

* **Initial dataset size**: ~500–1000 entries per functionality (queries, logs, YAML automations)
* **Update frequency**: Low for documentation; moderate for simulated logs and user sessions
* **Scalability**: Modular architecture allows for dataset expansion via synthetic generation or crowdsourced interactions

4. **Data Quality Requirements**

* **Completeness**: Logs should include full context (timestamps, status codes, device ID)
* **Clarity**: User queries must retain colloquial phrasing for real-world generalization
* **Schema Compliance**: YAML/JSON configs must validate against smart home schema or include comment annotations when malformed
* **Relevance**: Support docs must be device-specific with minimal noise or outdated instructions

5. **Potential Data Challenges**

* Variability in log structure between device manufacturers
* Lack of consistent terminology across brands (e.g., “routine,” “automation,” “scene”)
* Incomplete automation scripts sourced from GitHub
* Difficulty simulating multilingual queries without adding complexity

6. **Data Schema (Conceptual Layout)**

Here’s a sketch of your core schema—this can also be used in a diagram later:

{ user\_query: "Why won't my lights turn off at sunset?", device\_metadata: { brand: "Philips Hue", model: "Hue Bridge v2", firmware\_version: "1.93.7", connection: "Zigbee" }, config\_file: "automation.yaml", log\_excerpt: "No response from bridge. Timeout occurred.", support\_doc\_reference: "hue-troubleshooting.md", privacy\_policy\_snippet: "This device collects usage analytics..." }

This format should hit all the professor’s criteria: realistic scope, domain-specific accuracy, and alignment with your assistant’s core functionality. Up next, I can:

* Add this section to your PDF-style documentation
* Generate a data schema *visual* for your diagram section

A diagram of a process flow

AI-generated content may be incorrect.

📊 **Processing Pipeline Design** (Draft)

Your smart home assistant will handle a mix of structured, semi-structured, and unstructured data. Here’s the breakdown for the **data preparation pipeline**, with each stage documented for your report:

🧹 **Data Preprocessing Workflow**

| **Stage** | **Input** | **Operation** | **Output** |
| --- | --- | --- | --- |
| 1. Ingestion | Raw user queries, config files, logs | Load text/YAML/JSON from sources (manual entry, GitHub, public docs) | Raw datasets |
| 2. Text Cleaning | User queries, support docs | Normalize case, remove stopwords, tokenize, correct spelling | Cleaned text |
| 3. Metadata Normalization | Device data (brand/model info) | Map synonyms, standardize device fields, infer missing info | Structured metadata |
| 4. Config Validation | YAML/JSON files | Run schema checkers (e.g., PyYAML), extract automation rules | Parsed configs |
| 5. Log Parsing | Error logs | Use regex/line parsers to extract errors, timestamps, status codes | Diagnostic events |
| 6. Feature Construction | All cleaned inputs | Derive embeddings, tokenize configs, build user-context representations | Final feature set for retrieval |

🧠 **Feature Engineering Approach**

* **For user input:** Use embedding models (e.g., SentenceTransformers) to capture intent and context
* **For configurations:** Extract trigger-action patterns from YAML (when:, then: blocks)
* **For metadata:** Treat device type/brand as categorical features; normalize variants
* **For logs:** Derive frequency/timing of error messages and time deltas between events

**Processing Pipeline Design**

The SmartHome Assist system requires a robust, modular pipeline to handle diverse data types ranging from natural language queries to YAML-based automation configs and structured device metadata. This section outlines the end-to-end architecture for preparing such data for intelligent response generation.

**4.1 Data Preprocessing Workflow**

The assistant’s preprocessing pipeline is segmented into six major stages:

1. **Ingestion**: Raw data—including user queries, device logs, configuration files, and metadata—is collected from public repositories, user simulation sets, and open-source documentation. Sources include Home Assistant forums, GitHub YAML repos, and smart home vendor knowledge bases.
2. **Text Cleaning**: Natural language data (e.g., “Why won’t my lights turn off at sunset?”) undergoes lowercasing, tokenization, stopword removal, basic typo correction, and intent-preserving cleanup to retain conversational tone.
3. **Metadata Normalization**: Structured fields such as brand, model, or connection type are standardized using canonical mappings. For example, “Philips hue” → “Philips Hue” and “WiFi” → “802.11 Wireless.”
4. **Config Validation**: Automation scripts (YAML/JSON) are passed through schema checkers (using libraries like PyYAML or Cerberus). Parsed configurations are segmented into rule components (e.g., triggers, actions, conditions) for downstream use.
5. **Log Parsing**: Device logs are parsed using regular expressions or JSON loaders. Key fields such as timestamps, device IDs, and error codes are extracted to enable pattern recognition in fault resolution.
6. **Feature Construction**: All cleaned data is vectorized or encoded into model-compatible features. For instance:

* Queries and documents are embedded using pre-trained sentence transformers.
* Config actions are tokenized into labeled sequences.
* Metadata and logs are converted into tabular features or key-value formats.

The assistant’s preprocessing pipeline is structured into six key stages:

| **Stage** | **Input** | **Operation** | **Output** |
| --- | --- | --- | --- |
| **1. Ingestion** | Raw user queries, config files, logs | Load text/YAML/JSON from sources (manual entry, GitHub, public docs) | Raw datasets |
| **2. Text Cleaning** | User queries, support docs | Normalize case, remove stopwords, tokenize, correct spelling | Cleaned text |
| **3. Metadata Normalization** | Device metadata (brand, model info) | Map synonyms, standardize fields, infer missing info | Structured metadata |
| **4. Config Validation** | YAML/JSON config files | Run schema checkers (e.g., PyYAML), extract automation rules | Parsed configs |
| **5. Log Parsing** | Device logs | Use regex/line parsers to extract errors, timestamps, status codes | Diagnostic events |
| **6. Feature Construction** | All cleaned inputs | Create embeddings, tokenize configs, build context representations | Final features for retrieval |

**4.2 Feature Engineering Strategy**

The assistant’s intelligence hinges on meaningful feature representations:

**User Queries: Encoded via SentenceTransformers to capture intent and semantics for document matching.**

* **Automation Configs: YAML scripts parsed for triggers, conditions, and actions, converted to labeled sequences.**
* **Logs: Parsed into structured records capturing event frequency, time gaps, and status code patterns.**
* **Device Metadata: Categorical encoding of brand/model/firmware, supporting device-specific routing logic**

**4.3 Infrastructure Considerations**

The prototype will be built in **Google Colab**, leveraging lightweight tools like:

* spaCy and NLTK for text preprocessing
* PyYAML for config parsing
* Pandas and regex for log/metadata processing
* FAISS or ChromaDB for optional semantic retrieval

The architecture will support local testing with synthetic datasets and scale with additional modules as needed.

**5. Implementation Strategy**

The SmartHome Assist system will be developed as a lightweight AI assistant leveraging public datasets and open-source tooling. This section details the technical components, development plan, and team collaboration strategy.

**5.1 Technical Stack**

| **Component** | **Tool/Library** | **Purpose** |
| --- | --- | --- |
| **Text Preprocessing** | spaCy, NLTK | Tokenization, stopword removal, named entity detection |
| **YAML Validation & Parsing** | PyYAML, Cerberus | Schema checks and rule extraction from configs |
| **Log Parsing** | re (regex), Pandas | Extract events and normalize fields |
| **Vector Embedding & Retrieval** | SentenceTransformers, FAISS or ChromaDB | Intent matching and document similarity |
| **Development Environment** | Google Colab | Prototyping, visualization, and pipeline testing |

Optional expansion tools include Streamlit (for demo UI) or LangChain (if retrieval-based chaining is explored).

**5.3 Team Member Responsibilities**

| **Team Member** | **Role** |
| --- | --- |
| Fred (Example) | Lead Data Engineer: Preprocessing scripts, schema validation |
| Emmanuel | Feature Engineering & YAML parsing |
| Morgan | Embedding models and retrieval system |
| Jade | Documentation formatting and visuals (diagrams, PDF prep) |
| Monique |  |

Each member will cross-review another's section to ensure consistency and technical clarity.

**5.4 Resource Requirements**

* **Computational:** Google Colab (GPU not required), ~500MB storage per dataset batch
* **Data Storage:** GitHub repo or shared Google Drive for intermediate outputs
* **Human Resources:** 3–4 team members with overlapping skills in Python and data handling
* **Knowledge Sources:** Public docs (e.g., Home Assistant), smart home manuals, YAML config examples

**5.5 Implementation Challenges and Mitigations**

| **Challenge** | **Proposed Solution** |
| --- | --- |
| Inconsistent YAML formats | Add schema fallbacks and comment-aware parsers |
| Sparse or noisy logs | Use simulated data and apply pattern cleaning routines |
| Query phrasing diversity | Expand training queries with paraphrased variations |
| Data-to-response alignment for QA retrieval | Use cosine similarity and semantic rerankers |

**5.6 Integration Plan**

* The pipeline modules will be encapsulated as callable functions/notebooks and chained in a Colab-based prototype.
* Each component will pass structured outputs to downstream stages using shared Python objects or CSV/JSON files.
* The entire system will be modular, allowing for easy extension and upgrades during final project implementation

A diagram of a model training

AI-generated content may be incorrect.

**6. Evaluation Framework**

A rigorous evaluation framework is critical for assessing the effectiveness, reliability, and user value of SmartHome Assist. This section outlines how the system will be tested, validated, and improved over time.

**6.1 Success Metrics**

The assistant’s performance will be measured using a blend of quantitative and qualitative indicators:

| **Metric** | **Description** |
| --- | --- |
| **Intent Matching Accuracy** | % of user queries correctly mapped to relevant solutions |
| **Response Relevance Score** | Human-rated score (1–5) for each answer’s helpfulness and clarity |
| **Config/Log Parsing Coverage** | % of YAML/JSON configs and error logs successfully validated and extracted |
| **System Latency** | Average time to generate a response (in seconds) |

**6.2 Testing Strategy**

| **Component** | **Testing Method** |
| --- | --- |
| **Preprocessing Modules** | Unit tests with edge cases (e.g., malformed YAML, typos) |
| **Embedding & Retrieval** | Manual QA and automated semantic similarity checks |
| **Pipeline Integration** | End-to-end tests in Colab using predefined scenarios |
| **User Query Simulation** | Controlled test suite of synthetic queries and ground-truth matches |

A GitHub-based validation script repository will support reproducibility and tracking of performance over time.

**6.3 User Feedback Mechanism**

* User surveys and feedback forms (simulated or live) will be used to collect perceptions of clarity, accuracy, and ease of use.
* Suggestions and “Was this helpful?” prompts will support micro-feedback.
* Feedback examples may also be used as fine-tuning or augmentation data if extended beyond the midterm.

**6.4 Performance Benchmarks**

Minimum acceptable performance targets are:

* **80% accuracy** in intent classification on test set
* **Average relevance score ≥ 3.5** on a 5-point scale
* **Successful parsing of 90%+ YAML/log files** in sample set

These targets are designed to balance feasibility with impact, given the scope and tooling constraints.

**6.5 Continuous Improvement Plan**

* Iterative refinement of embeddings using new user queries and support content
* Expansion of synthetic data coverage and YAML rule library
* Periodic error analysis of logs/configs to improve edge-case handling
* Integration of updated device models and support pages every development sprint

CODING ------

# 📦 Imports

!pip install -q sentence-transformers PyYAML chromadb

from sentence\_transformers import SentenceTransformer, util

import yaml

import os

import json

import chromadb

from chromadb.config import Settings

import pandas as pd

# 🔍 Load a small test dataset (sample queries and rules)

sample\_docs = [

{

"id": "doc1",

"type": "yaml\_rule",

"content": "alias: Sunset Lights Off\ntrigger:\n - platform: sun\n event: sunset\naction:\n - service: light.turn\_off\n entity\_id: light.living\_room"

},

{

"id": "doc2",

"type": "faq",

"content": "If your lights do not turn off at sunset, verify that your location and time zone are configured correctly in Home Assistant settings."

}

]

sample\_queries = [

"Why won’t my living room lights turn off when the sun sets?",

"My sunset automation isn't triggering—what should I check?"

]

# 🤖 Load sentence transformer model

model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

# 🔗 Embed documents

doc\_embeddings = model.encode([d["content"] for d in sample\_docs], convert\_to\_tensor=True)

# ❓ Simple query interface

def retrieve\_response(query):

query\_embedding = model.encode(query, convert\_to\_tensor=True)

scores = util.cos\_sim(query\_embedding, doc\_embeddings)[0]

best\_idx = scores.argmax().item()

return sample\_docs[best\_idx]["content"]

# 🧪 Try it out

for query in sample\_queries:

print(f"\nUSER: {query}")

print("ASSISTANT:", retrieve\_response(query))

**Prototype Notebook**

# 🔧 STEP 1: Install Required Packages

!pip install -q sentence-transformers PyYAML chromadb

# 📦 STEP 2: Imports

from sentence\_transformers import SentenceTransformer, util

import yaml

import chromadb

from chromadb.config import Settings

import pandas as pd

import torch

# 🧠 STEP 3: Load Sentence Embedding Model

model = SentenceTransformer('paraphrase-MiniLM-L6-v2')

# 📚 STEP 4: Sample Docs – YAML Rule + FAQ Response

sample\_docs = [

{

"id": "doc1",

"type": "yaml\_rule",

"content": "alias: Sunset Lights Off\ntrigger:\n - platform: sun\n event: sunset\naction:\n - service: light.turn\_off\n entity\_id: light.living\_room"

},

{

"id": "doc2",

"type": "faq",

"content": "If your lights don’t turn off at sunset, check your automation's location and time zone settings in Home Assistant."

}

]

# 💬 STEP 5: Sample User Queries

sample\_queries = [

"Why won’t my living room lights turn off when the sun sets?",

"My sunset automation isn't triggering—what should I check?"

]

# 🔗 STEP 6: Embed Documents

doc\_texts = [doc["content"] for doc in sample\_docs]

doc\_embeddings = model.encode(doc\_texts, convert\_to\_tensor=True)

# 🔍 STEP 7: Define Simple Retrieval Function

def retrieve\_response(query):

query\_embedding = model.encode(query, convert\_to\_tensor=True)

scores = util.cos\_sim(query\_embedding, doc\_embeddings)[0]

best\_idx = torch.argmax(scores).item()

best\_doc = sample\_docs[best\_idx]

confidence = scores[best\_idx].item()

return f"🔎 Match (Confidence: {confidence:.2f}):\n\n{best\_doc['content']}"

# 🧪 STEP 8: Run MVP on Sample Queries

for q in sample\_queries:

print(f"\n🧑‍💻 USER: {q}")

print(retrieve\_response(q))