Overall Goal for the Month:

To develop a framework and architectural design for the Context-Aware Spatial Description (CAS-D) system and implement a functional MVP codebase that can process a video, manage contextual memory, and generate a description.

Week 1: Foundations & Environment Setup

Objective: Establish the theoretical basis and create the complete data processing pipeline and project structure, which is often the most time-consuming part of implementation.

- Theoretical Work:
    a. In-depth Paper Analysis: Deconstruct the core mechanisms of MC-ViT (non-parametric memory) and Video-3D LLM (position-aware representations). Diagram their data flows.
    b. Problem Formalization: Write a clear problem statement. Why is combining these two ideas novel and necessary for rich, contextual 3D scene description?
    c. Initial Architecture Sketch: Draw a rough, first-pass diagram of the CAS-D framework on a whiteboard or paper.

- Coding Work (MVP Focus: Data Pipeline):
    a. Environment Setup:
        i. Set up a Python virtual environment (e.g., Conda).
        ii. Install essential libraries: torch, torchvision, transformers, timm (for ViT models), numpy, and a library for handling 3D data if needed (e.g., open3d).
    b. Project Scaffolding: Create the basic file structure:
    c. Generated code (tentative structure)

```
casd_project/
├── data/              # For dataset files
├── src/
│   ├── dataset.py     # Data loading and preprocessing
│   ├── model.py       # The main CAS-D model architecture
│   ├── agent.py       # The memory consolidation agent
│   ├── train.py       # Training and evaluation loop
│   ├── config.py      # Hyperparameters and settings
├── notebooks/         # For exploration and visualization
├── README.md
    └── requirements.txt
```

    d. Dataset & DataLoader:

i. Download a suitable dataset. ScanNet is ideal because it provides RGB-D video streams with camera poses.
ii. In src/dataset.py, implement a PyTorch Dataset class that can:
    - Load a video sequence from the dataset.
    - For each frame, retrieve the RGB image, depth map, and camera intrinsic/extrinsic parameters.
    - Implement the "Maximum Coverage Sampling" from Video-3D LLM as a preprocessing step to select a fixed number of

Deliverable for Week 1:

- Theory: A document with diagrams and summaries of the key papers and a finalized problem statement.
- Code: A functional data pipeline. You should be able to run a script that loads a batch of sampled video frames and their corresponding 3D metadata, ready to be fed into a model. This is a huge milestone.

Week 2: Position-Aware Encoding

Objective: To implement the "eyes" of your system—the module that turns video frames into 3D-aware representations—and refine the system's architectural design.

- Theoretical Work:
    1. Formal Architecture Design: Turn your initial sketch into a formal architectural diagram with all modules, inputs, outputs, and tensor shapes clearly defined. This will be guided by your coding work.
    2. Mathematical Formalism: Write down the exact equations for creating the position-aware representation (e_vis = e_img + e_coord), including the 3D back-projection and the sinusoidal position encoding.

- Coding Work (MVP Focus: The Encoder):
    1. Implement the Encoder Module: In src/model.py, create a PositionAwareEncoder module.
        - Instantiate a pre-trained Vision Transformer (e.g., ViT-Base from timm or Hugging Face).
        - Write a helper function that takes a batch of depth maps and camera parameters and calculates the 3D world coordinates for the center of each patch.
        - Implement the sinusoidal position encoding function for the (x, y, z) coordinates.
        - Modify the forward pass:
            - Get patch embeddings (e_img) from the ViT.
            - Calculate coordinate embeddings (e_coord).
            - Return their sum (e_vis).
    2. Unit Testing: Write a simple test to ensure your encoder, given a dummy batch of data from your DataLoader, produces an output tensor of the correct shape (e.g., [batch_size, num_patches, embedding_dim]) without errors.

Deliverable for Week 2:

- Theory: A polished architectural diagram and the formalized mathematical definitions of your core components.
- Code: A functional PositionAwareEncoder module that is tested and integrated with your data pipeline.

Week 3: The Memory Consolidation Agent

Objective: To implement the "brain" of your system—the agent that manages memory—and design a robust evaluation strategy to prove its effectiveness.

- Theoretical Work:
    1. Agent Strategy: Justify the choice of memory consolidation. For the MVP, k-means is an excellent choice. Theorize why k-means centroids of position-aware tokens would be a powerful form of memory (e.g., they represent persistent objects or spatial regions).
    2. Evaluation Design: Design a specific, novel evaluation task for your framework. Create 5-10 example Question/Answer pairs that are impossible to answer without long-term context (e.g., "What object was on the table before the person put the laptop there?").
- Coding Work (MVP Focus: The Agent & Memory):
    1. Implement the Agent: In src/agent.py, create a MemoryAgent class.
        - Implement the consolidate method. For the MVP, this method will take a set of token embeddings (from a past video segment) and use a simple k-means algorithm (e.g., a PyTorch implementation or a wrapper around scikit-learn's) to find K centroids.

    2. Integrate Agent into Model: In src/model.py:
        - Instantiate the MemoryAgent.
        - Add a persistent buffer to your CASD_Model to act as the memory_bank.
        - Modify the main forward pass to process video in segments. For each segment, it should use the agent to update the memory_bank based on the previous segment's encodings.
    3. Decoder with Cross-Attention:
        - Choose a pre-trained decoder-only LLM (e.g., GPT-2, Llama).
        - In the forward pass, feed the decoder two things:
            1. The embeddings for the current segment (e_vis_t).
            2. The memory_bank as the encoder_hidden_states argument. This enables cross-attention.

Deliverable for Week 3:

- Theory: A document justifying the memory strategy and detailing the novel, context-dependent evaluation protocol.
- Code: A model that can process a video segment, update a memory bank via the k-means agent, and is wired for cross-attention.

Week 4: Integration, Training, and Final Proposal

Objective: To connect all components, run a proof-of-concept training loop, and synthesize all work into a final project proposal or paper draft.

- Theoretical Work:
    1. Write the Full Proposal/Paper Draft:
        - Write the Introduction and Related Work sections.
        - Write the Methodology section, using your diagrams and formalisms from previous weeks.
        - Write the Experiments section, detailing your dataset, the MVP implementation, and the proposed evaluation strategy.

- Coding Work (MVP Focus: End-to-End Run):
    1. Implement the Training Loop: In src/train.py:
        - Write a full training loop that iterates through your DataLoader.
        - In the loop, pass the data through your CASD_Model.
        - The model will output language logits from the decoder.
        - Compute the standard cross-entropy loss between the model's predictions and the ground-truth text descriptions.
        - Implement the backpropagation (loss.backward()) and optimizer step (optimizer.step()).
    2. Proof-of-Concept Run:
        - Run your train.py script on a very small subset of the data (e.g., 1-2 videos, batch size of 1) for a few dozen steps.
        - Goal: Verify that the code runs end-to-end without crashing and that the training loss decreases. This proves the entire architecture is viable.
    3. Code Finalization: Clean up the code, add comments, docstrings, and a comprehensive README.md explaining how to set up the environment and run the proof-of-concept.

Deliverable for Week 4:

- Theory: A polished, near-complete research proposal or paper draft.
- Code: A functional, end-to-end MVP codebase with a README.md. A successful demonstration shows the loss going down on a toy example.