ChatGPT

NVIDIA-Powered Architecture for Video Understanding and Conversational AI

Understanding an artistic film in depth requires a combination of computer vision, audio analysis, and language modeling. As an NVIDIA engineer, I recommend leveraging NVIDIA's latest vision and multimodal AI models in a **pre-processing pipeline** to produce rich, structured data about each film. This data can then feed into runtime conversational AI (chatbots) so they can have a realistic, context-aware discussion about the video as it plays. Below, I outline the challenges and a solution architecture using NVIDIA technologies, including how various models work in concert, whether fine-tuning is needed, and how to deploy the system.

Challenges in Analyzing Artistic Films

Films on escape.ai are highly varied: some are AI-generated or abstract visuals, some are live-action or animated stories, some have heavy dialogue while others have none. Key challenges include:

- **Visual Variety and Abstraction:** Scenes may be surreal or artistic, not just everyday objects. We need vision models that have broad perception and can describe abstract imagery or artistic styles.
- **No Predefined Labels:** Unlike traditional CV systems that detect a fixed set of objects, here we require open-ended understanding (e.g. describing **what is happening in a scene**, not just naming objects).
- **Audio and Dialogue:** If dialogue is present, transcribing it is crucial for context. If no dialogue, we rely purely on visuals.
- **Story and Deeper Meaning:** Beyond literal descriptions, we want to capture narrative elements, character interactions, mood, and even genre (is it comedic, dramatic, etc.). This requires higher-level reasoning over the sequence of scenes.
- **Temporal Continuity:** Understanding a film means linking events over time. We must track characters or objects across shots to know *who* does *what* with *whom* – important for identifying key interactions and continuity.
- **Real-Time Conversation Alignment:** Finally, the chatbots' commentary must sync with the video timeline, meaning our analysis needs time stamps and a way to fetch the relevant context for each moment.

NVIDIA Vision and Multimodal Models Overview

NVIDIA's latest AI models provide the foundation for tackling these challenges. **Vision-Language Models (VLMs)** are a new class of generative models that combine visual understanding with language, far exceeding the capabilities of traditional CV alone. Unlike a fixed object detector, a VLM can produce rich *textual descriptions* of images or video frames and even answer questions about visual content. This broader perception and contextual understanding is crucial for artistic films [1]. For example, a VLM can describe an abstract scene ("a swirling pattern of blue and purple lights") or interpret complex frames in natural

language. NVIDIA's VLMs (such as the VILA 1.5 model family) are designed to run efficiently on GPUs and support tasks like image captioning, object detection, and even segmentation in one framework [2] [3].

However, a single VLM can only process a limited number of frames at once and has a finite context window [4]. Long videos can easily exceed these limits, and important details might be missed if frames are sampled too sparsely [5]. Additionally, pure visual models don't "hear" audio – without audio transcription, they miss spoken context [1]. To overcome these limits, NVIDIA provides a **Blueprint for Video Search and Summarization (VSS)** that *combines* VLMs with Large Language Models (LLMs) and Retrieval-Augmented Generation (RAG) techniques [6]. In essence, the solution is to break the video into manageable pieces, analyze each piece with vision (and audio) models, then use an LLM to aggregate and reason over the pieces. The blueprint implements a full pipeline for stored or real-time video analysis, enabling long-form video understanding with summaries, Q&A, and more [6] [7].

Large Language Models (LLMs) (like the latest Llama or GPT-based models) handle the higher-level reasoning: summarizing the film, inferring context or themes, and engaging in conversation. NVIDIA's NeMo framework and NGC catalog provide many LLM options (for example, a 70B-parameter Llama 3.1 model is used in the NVIDIA blueprint for summarization and Q&A) [8]. These models can be accessed via the NVIDIA API Catalog using a REST API key, which fits your use-case of using NVIDIA's hosted services for now [8].

Audio Models (ASR & TTS): For films with dialogue or narration, NVIDIA Riva Speech AI is ideal. Riva's Automatic Speech Recognition (ASR) model (available as a NeMo Microservice) can produce accurate transcripts of spoken audio in real time [9]. Incorporating speech-to-text *greatly enhances* scene understanding by adding a multimodal perspective [10] [11]. For example, a VLM might see two characters moving but the dialogue tells us *why* (plot motivations or jokes). By converting speech to text, we enable the LLM to factor in both visual and audio information for full context [12]. Riva also provides text-to-speech (TTS) for output. While TTS isn't needed for analysis, it becomes important if our chatbots will speak aloud (more on this in the deployment section).

In summary, we will use **specialized models for each modality**: an ASR for audio, a VLM for visual frames, and an LLM for reasoning and language generation. Each excels at its task – for instance, Riva ASR will outperform a generic multimodal model in transcription accuracy, and a dedicated VLM will provide better image descriptions than an LLM with no visual input. The key is orchestrating them properly to produce a cohesive understanding of the film.

Pre-Processing Pipeline Architecture

To extract all necessary knowledge from a film, I propose a pipeline with multiple stages. This largely mirrors the NVIDIA Video Search & Summarization blueprint (adapted to artistic content). The major components are illustrated in **Figure 1** and described below.

Figure 1: High-level architecture of the video analysis pipeline, combining audio transcription, computer vision, and language modeling. Video frames and audio are processed in chunks, generating transcripts, object metadata, and dense captions for each segment. These are stored in a vector database and a knowledge graph [13] [14]. An LLM then uses retrieval (RAG) and the graph to produce summaries or answer questions about the whole video.

1. Speech-to-Text Transcription (Audio Processing)

If the film contains any spoken dialogue or narration, the first step is to extract it as text. We split the audio track from the video and feed it through NVIDIA's ASR service (part of Riva). This service (accessible via REST with an API key) will return a timestamped transcript [15]. The blueprint performs this by chunking the audio (in sync with video chunks) and invoking the **Riva ASR** microservice to get text for each segment [15]. The result is a sequence of text snippets with timecodes, capturing who said what (the model won't know character names, but can separate speakers if acoustic models support diarization).

The transcription is crucial for multi-modal understanding – it provides additional context to the visual analysis. For instance, if a scene has no obvious action but the dialogue is comedic, the transcript would reveal the humor, helping the LLM infer that the genre or tone is comedic. The blueprint notes that adding "the ability to hear" improved contextual understanding significantly [16] [12]. We will combine these transcripts with visual data in later steps.

(Under the hood, Riva ASR uses state-of-the-art Conformer models for speech recognition, and can be customized for specific domains or accents if needed. However, for initial deployment, the general English model should suffice, with support for other languages if your films include them [9].)

2. Visual Scene Analysis with Vision-Language Models

In parallel with audio, we process the video frames to understand the visual content. Rather than using a simple object detector, we use an NVIDIA **Vision-Language Model (VLM)** to generate **dense captions** and embeddings for video segments. "Dense captions" means a descriptive summary of what is seen in a short clip (e.g. *"A woman in a red dress is dancing on an empty stage while a blue spotlight moves across her"*). These captions go beyond isolated object labels, providing a narrative of each chunk of video.

Chunking: To handle long videos, we divide the film into chunks of a certain duration (configurable, say a few seconds each) [17] [18]. For each chunk, we sample a set of frames (the VLM might take e.g. 8 or 16 frames as input at a time). The NVIDIA blueprint uses a sliding window with overlap so that events straddling chunk boundaries are still caught [17] [19]. This ensures, for example, if a critical action happens at the border of two chunks, it won't be entirely missed.

VLM Inference: Each chunk's frames are fed to the visual encoder (part of the VLM model). NVIDIA's VLM (code-named **NVIL/VILA**) encodes images into embeddings, then uses a language model to produce a text description [20] [21]. This is effectively like running an image captioning model on the frames, but optimized for multiple frames to catch actions. The output per chunk might be a few sentences describing everything of note in that interval. We can also prompt the VLM to focus on particular aspects. For instance, the blueprint allows a *VLM prompt* to specify what objects or events to pay attention to [17] [22]. In an artistic film context, we might prompt the VLM with something like: *"Describe the people, animals, important objects, and any notable actions or scene changes in detail."* This guiding prompt (and the **Set-of-Mark (SoM)** mechanism) helps the model not overlook key elements [23]. NVIDIA's updated CV pipeline uses SoM with **zero-shot object detection** to improve VLM focus on specific labels [10] – essentially, we can provide a list of things we care about (e.g. "human figures, faces, text on screen, specific symbols…") and the VLM will be guided to detect and describe those [23].

Object Detection & Tracking (CV Metadata): In addition to free-form captions, we may want structured metadata about objects and characters. The pipeline can integrate NVIDIA's **DeepStream** SDK to run a CV sub-pipeline on the frames [24]. This would perform: (a) **Object detection** on each chunk (using a model like Grounding DINO for zero-shot detection with text labels) [25], (b) **Instance segmentation** (using Meta's SAM, integrated via DeepStream) to get object masks [26], and (c) **Object tracking** across frames (using the NvDCF tracker in DeepStream) to assign persistent IDs to objects/persons across the chunk [27]. The outcome is that for each chunk we know, for example, Object #5 is "a man in a hat" and Object #7 is "a dog", and we have their bounding boxes or masks over time [25] [28]. Crucially, the pipeline *fuses* metadata across chunks so that if the same object reappears in a later segment, it can reconcile the IDs [29]. This fusion gives a continuous identity for characters even if they disappear and reappear [29].

Why is this important? Because it lets the VLM and later the LLM talk about *specific characters consistently*. The blueprint uses this to allow queries like "Which car ran the red light?" and the system knows which exact car (ID 21, say) was involved across frames [30] [31]. In a film, this means we could label "MainCharacter (ID 1)" and "Villain (ID 2)" etc., enabling the LLM to understand their interactions over time. The VLM can incorporate these tags when generating captions (so instead of "a person does X" it can say "Person [ID#1] does X"), which yields more precise and referable descriptions [32] [31]. For artistic cases, if the film has recurring visual motifs or characters, this tracking is very useful to capture interactions (e.g. *"[ID2] approaches [ID1] and hands them a glowing orb"* – we know it's an interaction between two specific characters).

Output of Visual Analysis: For each chunk, we now have: a dense caption text, plus structured metadata (objects with labels, positions, and possibly who is interacting with whom). These, along with the chunk's timestamp range, form the **visual understanding** of the film.

3. Data Storage and Indexing (RAG Preparation)

All the outputs from audio and visual analysis are then stored for downstream retrieval. NVIDIA's blueprint uses two types of databases in the Retrieval-Augmented Generation stage:

- **Vector Database (Embedding Store):** Each chunk's caption (and possibly transcript segment) is converted into an embedding (vector) and stored in a vector DB (like Milvus). This allows semantic search over the video content [33] [34]. For example, if later the chatbot or user asks "What was the protagonist doing in the first scene?", the system can vector-search for relevant chunks about "protagonist" and "first scene" to fetch the right data.
- **Knowledge Graph (Graph Database):** In addition to free text, the blueprint creates a **knowledge graph** of the video [35] [36]. Nodes in this graph might represent entities (characters, objects, locations) and events, with edges denoting relationships (e.g. *"Character_A –[gave]→ Object_X"* or *"Character_A –[met]→ Character_B in Scene_3"*). The dense captions can be parsed by an LLM into structured triples and loaded into the graph DB [36]. This graph is extremely useful for complex querying and maintaining story continuity – it encodes the "who, what, when, where" of the film in a connected form. The Graph-RAG module in NVIDIA's pipeline populates this graph and allows an LLM to query it when answering questions [35]. In our case, the graph could capture character interactions (nodes for each character and an edge for each interaction event with a timestamp). This directly addresses the need to detect "specific key frames where character interactions happen" – those would become edges or events in the graph (for example, a node "Meeting" connected to

two character nodes at time T). We will use the graph to ensure the chatbots remember relationships and prior events as they converse.

After ingestion, we have a **comprehensive index** of the video in text form: transcripts, per-scene descriptions, object metadata, and a knowledge graph of entities/events [37]. At this stage, the heavy lifting by the perception models is done. The data is now ready to be fed into generative AI models for summarization and conversation.

(Note: The NVIDIA blueprint provides a ready deployment of this whole ingestion pipeline, with REST APIs like `/summarize` *and* `/chat/completions` *to interact with the processed data [38] [39]. You could potentially use the blueprint as-is, pointing it at your video files, and it will handle chunking, model calls, and building the index. For learning and customization, though, it's useful to understand and possibly implement each step as described.)*

4. Summarization and Contextual Analysis with LLM (CA-RAG)

Once the video's data is indexed, we use a **Large Language Model** to synthesize higher-level insights. This happens in a couple of stages:

- **Chunk Summarization:** An LLM can be prompted (via a *Context-Aware RAG* module) to aggregate the *dense captions* from all chunks into a cohesive summary [33] [40]. Essentially, it reads the sequence of chunk descriptions and writes a more concise narrative of the film. NVIDIA's blueprint does this automatically after ingestion – it feeds the ordered captions into an LLM with a prompt like "Summarize the video content". The result might be a paragraph or two covering the major events and timeline of the film, which is extremely useful for quick understanding. We can control the style and detail via prompt (e.g., whether we want a brief summary or a detailed scene-by-scene recap) [41]. This summary is one artifact we'll provide to the chatbots.

- **Thematic Analysis and Genre:** We can further prompt the LLM (using the film's transcripts + summary as input) to answer questions like "What is the overall theme or message of this film?", "How would you describe the genre (comedy, drama, etc.)?", or "What emotions does the film evoke?". Because the LLM has knowledge of story conventions and tone, it can infer these higher-level attributes. For example, it might detect irony or humor in the dialogue and label the film a dark comedy. It might recognize an abstract visual metaphor and hypothesize a deeper meaning. These are subjective outputs, but a powerful LLM can provide useful interpretations. We don't necessarily need a dedicated model for genre classification; a prompt to the LLM with instructions to identify style and genre should suffice (and we can double-check its answer for reasonableness).

- **Knowledge Graph QA:** For fine-grained details or to verify consistency, the LLM can query the knowledge graph. NVIDIA's Graph-RAG allows the LLM to retrieve specific nodes/edges related to a question [36]. For instance, if asked "When do the two main characters first meet?", the LLM could search the graph for an edge connecting those two characters and find the timestamp/scene. This ensures the chatbots can get precise answers about the film's events when needed (and not hallucinate or rely solely on memory). The graph essentially acts as a fact-checker and memory for the LLM [36] [37].

- **RAG for Long Conversations:** Since our chatbots will converse possibly for the length of the film, we'll use Retrieval-Augmented Generation (RAG) to keep their knowledge fresh. This means at any

point, the bots can pull in the relevant chunk descriptions or transcripts from the vector store given the current timestamp or topic. This prevents the LLM from forgetting earlier scenes or introducing inconsistencies, as it can always retrieve the ground-truth context from the index when formulating responses [37].

By the end of this phase, we have multiple artifacts: detailed per-scene captions, a timeline of events, an overall summary, transcripts, and a knowledge graph of characters and interactions. **All of this was achieved with pre-trained models and smart orchestration – no model has been fine-tuned on the specific film, we've just *inserted* the film's data into the models' workflow (via RAG).** This is powerful because it means we can process any new film without retraining, simply by indexing its content and letting the LLM reason over it.

Model Selection and Roles

Now, to answer the specific question of which models to use for what task (and whether fine-tuning is needed):

- **Vision-Language Model:** Use NVIDIA's latest VLM (for example, the **Llama-3.1-Nemotron-VL** model, which combines a Llama 3.1 language backbone with a vision encoder) [20] [21]. This model is available through NVIDIA's NeMo service/API Catalog, so you can call it with an API key. It supports image and video inputs and can handle tasks like image Q&A, captioning, and OCR [20] [42]. For our use, we'll primarily use its captioning ability on video frames. If needed, this model (or variants of VILA) comes in different sizes (some optimized for speed vs. accuracy) [3], so you can choose a smaller one if latency or single-GPU deployment is a concern. The blog mentions a Nano 8B version for edge devices and larger ones for data center [3]. On AWS with a capable GPU, you might opt for the larger model to get better captions. Importantly, **fine-tuning the VLM is usually** not **required** – these models are trained on diverse image/video data, so they can describe most content reasonably well out-of-the-box. If your films have a very unique visual style where the VLM consistently fails, you could consider fine-tuning or prompt-tuning it, but try zero-shot first. Often, giving a good prompt or using the SoM labeling trick is enough to get relevant outputs [41]. Fine-tuning a vision model would also demand a lot of example frames from your domain, which may not be available.

- **Automatic Speech Recognition:** Use **NVIDIA Riva ASR** (which internally might use the Citrinet or Conformer models) via the NeMo API. This will handle English and several other languages with high accuracy [12]. Unless your films have very domain-specific vocabulary (in which case you could customize the language model or add a custom dictionary to Riva), there's no need to train an ASR model from scratch. Riva is also real-time if needed, which suits streaming. We feed it chunks of audio as described earlier. No fine-tuning here in most cases – just ensure the acoustic model supports the language spoken. Riva can be adapted with additional training data if you have a collection of film audio and transcripts (this would be an ASR adaptation, not something like an LLM fine-tune).

- **Object Detection/Tracking (Optional):** If you decide to implement the CV enhancement pipeline, you'd use models like **Grounding DINO** (available on NGC) for detection and **SAM (Segment Anything)** for segmentation masks [25] [26]. These are state-of-the-art models released by Meta and integrated in NVIDIA's samples. You don't train these from scratch; just use them as-is to get zero-shot detection for arbitrary classes. If your use-case needed to detect a very specific visual concept

(say a particular logo or art style), then using NVIDIA's TAO Toolkit to fine-tune a classification or detection model *could* be an option. For example, TAO could train a custom ResNet or YOLO model to detect a recurring symbol that the generic models might not recognize [43] [44]. But for most film understanding tasks, the combination of VLM captions plus a zero-shot detector covers enough. TAO is more relevant if you have a predefined set of objects/actions you want to monitor (common in surveillance or industrial video analytics), whereas here we want open-ended descriptions. So I'd lean on the foundation models rather than custom-trained models, unless a particular need arises.

- **Large Language Model:** Use a strong **LLM** for summarization and conversation. NVIDIA's blueprint example uses *Llama-3.1 70B Instruct*, which is a good choice [8]. NVIDIA offers it as a hosted model (you can call it via API), or you could deploy it on a GPU server if available. There are also smaller LLMs (e.g. 8B or 20B sizes) that could run on a single GPU with lower memory, if needed [45] [46] – though with smaller models, the quality of the conversation may suffer, especially in grasping nuance from the film. Given that conversation quality is paramount, I'd recommend using the largest model you can reasonably call or host. **Fine-tuning the LLM** is likely **not needed** for this task either. Retrieval-Augmented Generation means we feed the model with all necessary info about the film at inference time, so the model doesn't need to have seen the film in training. You might consider fine-tuning or prompt-tuning the LLM *for dialogue style* if you have specific requirements (for example, making the chatbots mimic a certain personality or speak in a certain tone consistently). Even that can often be achieved with clever prompting (system prompts that establish the persona). NVIDIA NeMo Toolbox allows fine-tuning LLMs on custom data, but it's a heavy process for 70B models. A lighter alternative is using **NeMo Guardrails** and prompt templates to guide the LLM's behavior (ensuring it stays on topic, avoids disallowed content, etc.) [47] [48]. Guardrails could ensure the bots only talk about the film and not something random, for instance.

- **Retrieval and Databases:** For RAG, you'll need a vector database and possibly a graph database. These aren't AI models, but worth mentioning tools: Milvus or Faiss for vector search, and Neo4j or an in-memory python graph for knowledge graph. NVIDIA's stack uses Milvus and Neo4j in their examples. No training here, just deployment and data loading.

- **Orchestration and Streaming:** NVIDIA **DeepStream SDK** is useful to efficiently implement the video chunking, decoding, and CV pipeline on GPU [26] [49]. If you want to analyze videos in *near-real-time* (or many videos in parallel), DeepStream provides a C++/Python framework to pipeline decode → inference → tracking on the GPU with minimal overhead. It's not a model itself but an SDK. In our architecture, DeepStream could be the backbone that feeds frames to the VLM model and coordinates with the ASR. The blueprint indeed bases its ingestion on DeepStream for handling video input and concurrency [24]. Since you might deploy on AWS, you could run an NVIDIA DeepStream Docker container with your models and code inside for scalable processing. If using the cloud API route (NVIDIA's hosted models), you might not need to use DeepStream explicitly – you can write a Python script that reads the video, chops into frames, and calls the API for each chunk. But for production and speed, DeepStream or NVIDIA's **NIM** microservice infrastructure is the way to go (NIM = NVIDIA AI Microservices, which is what those API endpoints are under the hood).

- **NVIDIA TAO Toolkit:** As mentioned, TAO is there if you need to fine-tune or train a custom vision model. For example, if you have a series of films with a recurring animated character and the generic models don't reliably recognize that character, you could use TAO to train a classifier to tag frames where that character appears, using a few hundred labeled images. TAO provides pretrained model

weights and easy fine-tuning pipelines for vision tasks [50] [44]. After TAO training, you'd integrate that model into the pipeline (perhaps as an added detection step). But I would treat this as an optional enhancement only if required – start with the foundation models' output first.

In summary, **each model has a clear role**: Riva ASR for speech-to-text, VLM (e.g. NVIDIA VILA/Nemotron) for visual understanding, DeepStream modules for tracking and efficiency, and a powerful LLM (Llama-based) for summarization and conversation. The models can be used in sequence (first ASR/VLM, then LLM) or concurrently as agents that pass data, but ultimately they form one coherent workflow.

Workflow: Putting It All Together

Let's walk through how you might execute this pipeline step-by-step, and consider the sequence and any agent-to-agent orchestration:

1. **Ingestion Phase (Offline or Pre-processing):** For a given film, automate the process of extracting audio, chopping video into frames/chunks, and calling the respective models. This could be done with a script or with an orchestrator like the NVIDIA NeMo Agent Toolkit (Agentic SDK). For instance, Agent Toolkit can coordinate a vision agent and an audio agent to work in parallel, and then trigger a summarization agent [51] [52]. The key is to ensure the results (captions, transcripts) are time-stamped and stored.
2. You might run the ASR and VLM on each chunk in parallel threads, or do audio first then video – the order isn't crucial as long as both modalities get processed. In practice, doing them in parallel saves time.
3. After chunk processing, a **retrieval/indexing agent** (or just a function) stores the outputs into the vector DB and graph DB.
4. Then a **summary agent** (LLM) is invoked to produce the overall summary and maybe other analyses (like extracting a list of characters, identifying the setting, etc.). These can also be stored or just kept for use.

5. NVIDIA's pipeline can run this ingestion on multiple GPUs or even multiple videos concurrently if needed [53] [54], but for one film at a time, a single GPU can handle it. They even have a single-GPU mode for the blueprint using a smaller model combination [45].

6. **Runtime Phase (Live Conversation):** Now the film is about to play for an audience with the chatbots active. At runtime, we do **not** want to redo heavy analysis; instead we use the prepared data. The chatbots (which are essentially instances of the LLM with assigned personas) will generate dialogue turn by turn. To keep them in sync with the film:

7. We use the timestamp to fetch context from our index. For example, if the film is at 1:02 (mm:ss), we know this falls in chunk 5 (say) which corresponds to a certain scene. We retrieve the caption for chunk 5, any transcript lines around 1:00–1:05, and relevant graph info (e.g. which characters are present in this scene). This context is fed into the prompt for the chatbots.
8. The bots' prompt could be structured like: *System prompt:* "You are film characters/commentators discussing the film as it plays. Stay in character and only talk about events happening on screen or previously occurred." Then *Context:* (here we insert the caption: e.g. "Scene: [ID2] hands [ID1] a glowing orb. [ID1] says: 'This is our key to freedom.' [ID2] smiles." plus any relevant backstory from earlier scenes). Then *Conversation history:* the previous few chatbot exchanges (if any).

9. With this prompt, the LLM (chatbot agent) generates the next line or two of dialogue for each character or each bot. Because we gave it the context, it will speak relevantly about the current scene (e.g. "Bot1: *amazed tone* I can't believe she finally got the orb – this changes everything!").
10. We alternate between the bots: each one gets the updated context and possibly the last line from the other, and produces a response. This creates a back-and-forth conversation that is grounded in the scene.
11. The use of RAG here is crucial: the bots can "remember" earlier scenes by querying the vector DB or graph for past events if they need to reference them (for instance, if Bot2 wants to recall "As we saw in the beginning, she was hesitant to take the orb," it can retrieve that info from the knowledge graph where the initial refusal event was recorded). This ensures continuity and realism in the conversation.
12. We might also incorporate **NeMo Guardrails** to filter or adjust the bots' outputs – e.g., to avoid spoilers (if that's a concern) or prevent any offensive language if the film content is sensitive [47] [55]. Guardrails could also enforce timing (making sure each bot response is brief enough not to lag behind the scene).

Importantly, we do **not** need to run the heavy VLM or ASR models during runtime playback (unless you want the system to handle totally unseen live video). Since the film is known and preprocessed, the chatbots can rely on the stored descriptions. This makes the runtime much lighter: essentially just LLM inference and database lookups, which a single GPU or even CPU (for the DB) can handle. The chat experience will be far more efficient this way, as opposed to trying to have an AI "watch" the video live frame-by-frame. We only need to ensure the clock syncing mechanism fetches the right context at the right time.

1. **Output to Users (Avatar/Voice):** Once the chatbot text lines are generated, you can output them as you see fit. If it's just text on screen, that's straightforward. If you want them spoken by avatars (MetaHumans or NVIDIA's avatars), you'll bring TTS and animation into play. NVIDIA **ACE (Avatar Cloud Engine)** is a toolkit that combines these pieces to create interactive digital characters. In your case, the bots aren't exactly interactive with the user, but ACE can still help connect the speech and animation:
2. Use **Riva TTS** (part of ACE) to synthesize each bot's lines into speech audio. Riva has multiple voices, or you can integrate a third-party TTS if preferred. The TTS will output an audio stream for each sentence nearly in real time (few hundred milliseconds typically).
3. If you use Unreal Engine MetaHumans, you can use Unreal's facial animation pipeline or NVIDIA's **Audio2Face** to drive the character's lip-sync from the audio. NVIDIA actually provides a plugin for Unreal Engine (as part of ACE) that takes the Riva TTS audio and animates a MetaHuman's face **in real time** [56] [57]. This significantly simplifies tying the voice to the character's mouth movements. There are ACE demos showing exactly this: a character speaks with Riva voice and is animated via Audio2Face, all synced in UE5.
4. If using NVIDIA's own avatar solution (e.g., Toy Jensen demo or others), ACE would similarly handle it – but given you are leaning toward MetaHumans, the above approach stands.
5. We don't need any *vision* model in this output stage; it's purely using the precomputed text to generate speech and animation.

In essence, **one could deploy the entire pipeline** such that: the video plays in sync with a timeline; at each interval, the system retrieves context and generates chatbot dialogue; that dialogue is then spoken by animated characters either in-engine or via a streaming pipeline. The heavy analysis (VLM, ASR) is all done beforehand or on a separate GPU service, so the conversation flows without hiccups.

Fine-Tuning vs. RAG: What's Necessary?

Your question specifically asks if fine-tuning any models on the film data is required, or if a Retrieval-Augmented Generation approach is sufficient. Based on the design above, **RAG is not only sufficient but highly recommended over fine-tuning** for this scenario. Here's why:

- **Scope of Data:** Each film is a relatively small dataset (a single sequence of scenes). Fine-tuning a large model on one film's data would risk overfitting and wouldn't generalize to the next film. It's also costly and time-consuming, whereas RAG lets us use the film data at inference time directly.
- **Use of Foundation Models:** NVIDIA's foundation models (VLM, LLM) are trained on broad data and already *know* how to see and talk. We just need to give them the right information (via prompts and retrieval). This is exactly the purpose of RAG – it "augments" the model's knowledge with the specific details from our film [6]. The blueprint architecture is built on the premise that you do **not** fine-tune the LLM on each video; you store the video's content and let the LLM retrieve it as needed [40] [37].
- **Customization:** That said, there are a few niche cases where fine-tuning or additional training is useful:
- If you wanted the VLM to better handle *artistic images* specifically, you could fine-tune it on a dataset of similar images with captions. But current models (like those based on LAION or COCO data) already have seen a lot of art and abstract images, so they often do surprisingly well even on surreal content.
- If the conversation style of the chatbots needed to mimic a particular script or characters (say you want them to speak in Shakespearean English or in the style of a famous director), you might fine-tune or at least **few-shot prompt** the LLM on example dialogues. A light fine-tune (or LoRA adapter) on conversation data could make it more consistent in persona. However, an easier route is to write a good system prompt defining each bot's personality.
- If the ASR struggles with the audio (e.g., heavy background music, very stylized speech), you might gather some transcribed audio from the films and fine-tune the acoustic model or language model of Riva. Riva allows custom language model integration (for jargon or accents) [58] [59]. This is a relatively small effort compared to LLM fine-tuning.
- If certain visual motifs are completely missed by the VLM (for example, maybe the films have hidden QR codes or text that the VLM doesn't read), you might bring in an OCR model or train one to pick that up. But again, many VLMs (like NVIDIA's) include OCR ability inherently [42].

In summary, **no core model in this pipeline strictly requires fine-tuning on your film data.** The strategy is to use pre-trained models + metadata extraction (RAG) to adapt to each film. This is faster and more scalable as you add more films. Fine-tuning might come into play as an optimization down the road, once you have a lot of usage data and see specific weaknesses in the pipeline.

Deployment Considerations

You mentioned deploying on AWS or Cloudflare and using NVIDIA's REST APIs initially. Here's how that can be achieved:

- **NVIDIA REST APIs (NeMo NGC):** NVIDIA offers hosted endpoints for many models through the **NGC API Catalog** [8]. You can obtain an API key and call these services over HTTPS. For example, there are endpoints for `riva/asr` (for speech-to-text), for `nvidia/llama-3.1-70b` (LLM text generation), and for vision models like `nvidia/llama-3.1-nemotron-nano-vl-8b` (the

multimodal model we discussed) [60] [61]. Using these means you don't need to host models yourself during prototyping – you just send requests with your data and get back results. Do note the latency and cost considerations: sending many frames to a VLM or many prompts to an LLM over the internet will have some latency. However, since this is a preprocessing step, it might be fine if it takes a minute or two to analyze a short film. For live chatbot responses, the LLM API latency (likely a few seconds per turn) will determine how fast the conversation is. If that's too slow, you might eventually deploy a local LLM for real-time needs, or use a smaller model.

- **AWS Deployment:** If you choose to host models yourself on AWS, you'll want GPU instances (NVIDIA A10G, A100, or newer H100 if available on cloud). You could run the entire pipeline in a container (NVIDIA provides Docker images for Riva, DeepStream, and the NeMo frameworks). For instance, you could run an **Riva server** on the instance to handle ASR/TTS. You could load a **Triton Inference Server** for the VLM model (Triton is NVIDIA's serving software that supports multimodal models too). And you could run the LLM in another container or use the API still. This is more complex initially, which is why using the API for now is a good idea. The blueprint itself is available as Docker Compose scripts that spin up all needed microservices (VLM, LLM, DBs, etc.) – you might be able to deploy that on AWS for an all-in-one solution [62] [63].
- **Cloudflare Deployment:** Cloudflare is a bit trickier for heavy AI workloads – their workers are not GPU-based. However, Cloudflare could be used as a proxy or coordination layer (for example, a Worker could handle incoming requests and orchestrate calls to the NVIDIA APIs or to an AWS backend). If Cloudflare has an AI inferencing service (they've talked about AI at the edge, but generally not with large models yet), it might not handle this level of processing. So likely, your actual AI processing will live on an NVIDIA GPU somewhere (either your own or NVIDIA's cloud), and Cloudflare could host the web application or timing logic.

- **Scaling and Real-Time:** If you eventually want to handle live streams or multiple films concurrently (like multiple users watching different films with their own chatbot commentary), the multi-stream capability of the blueprint can help [64] [65]. It can maintain separate contexts per stream and even run them in parallel on one system. The design we've discussed is fully capable of real-time analysis too (with powerful GPUs), but since we can precompute, we avoid the need for ultra low-latency vision processing at runtime. We essentially trade storage for compute: store the analysis results, then stream those to the LLM as needed.

- **NeMo Agent Toolkit:** Since you mentioned NVIDIA's "agentic programming SDK," that likely refers to the NeMo Agent Toolkit (formerly called Agent IQ) [66]. This is an optional layer that provides **monitoring and optimization for multi-agent systems**. If you have multiple bots and multiple tools (like a retrieval tool, a calculation tool, etc.), Agent Toolkit helps profile where time is spent, parallelize calls, and manage tool usage across agents [51] [52]. In our context, you could use it to manage the workflow of (VisionAgent → DB → LLMAgent). It's especially useful if you extend the system with more complex decision logic. For a simpler pipeline, you might not need it, but it's good to know it exists for scaling up. It also integrates with LangChain, so if you use LangChain to implement the RAG logic and agent conversation, Agent Toolkit can instrument that to ensure efficiency [67] [68].

- **Monitoring and Improvement:** Once deployed, you'd gather data on how well the chatbots perform. NVIDIA's tools can help here too. For example, you can log each agent's inputs/outputs and see if they made mistakes in understanding. If the LLM consistently misunderstands a particular visual, you might refine the prompt or add a custom detection for that case. The modular nature of

the pipeline makes such adjustments easier – you can upgrade one component (say, swap in a better VLM when available, or fine-tune a model via TAO) without overhauling everything.

Conclusion

By combining NVIDIA's cutting-edge vision models, speech AI, and large language models, we can build a system that truly **understands** an artistic film and enables rich, in-time commentary. In the preprocessing phase, the film is ingested and analyzed from every angle: *what is seen* (via VLM dense captions and CV metadata) and *what is said* (via ASR transcripts), yielding a structured representation of the film's content. This representation (stored as embeddings, summaries, and graphs) is then used to prompt an LLM-driven chatbot (or multiple chatbots) that can discuss the film with awareness of context, characters, and plot. The chatbot conversation can be made even more engaging by deploying it through NVIDIA ACE – giving each AI a voice and an animated persona that can be synchronized with the film.

Crucially, this architecture leans on **Retrieval-Augmented Generation (RAG)** rather than training custom models for each film, which makes it general and scalable. Each model in the pipeline is used for the task it's best at (vision, speech, or language), and NVIDIA's framework ties them together. The NVIDIA Video Analytics AI Blueprint essentially provides a reference for this kind of system, and its recent GA release includes features like audio transcription, object tracking with SoM, and GraphRAG that align perfectly with your needs [10] [23].

In practical terms, you can start by calling NVIDIA's hosted APIs (for ASR, VLM, LLM) with your film data to prototype the pipeline. This will give you immediate insight into how well the models describe your content. From there, you can deploy the components on AWS for more control, and integrate with your front-end (Unreal MetaHumans or otherwise) for the final user-facing experience. The result will be an AI-powered commentary system that watches the film "like a human," understanding scenes and discussing them with nuance and coherence.

Sources: The design above is informed by NVIDIA's technical blogs on multimodal video agents and the capabilities of the NVIDIA AI stack for vision and conversational AI [1] [69] [25] [36] [9], which demonstrate how VLMs, LLMs, Riva, DeepStream, and RAG techniques can be combined to achieve deep video understanding and interactive Q&A or summarization. All these technologies are available through NVIDIA's developer platform and can be adapted to the artistic film domain with minimal or no fine-tuning, as discussed. The key is careful orchestration and leveraging each tool for its strength – which is exactly what we've planned in this system design.

---

[1] [6] [9] [10] [11] [12] [15] [16] [23] [25] [26] [27] [28] [29] [30] [31] [32] [34] [38] [39] [45] [46] [49] [53] [54] [58] [59] [62] [63] [64] [65] Advance Video Analytics AI Agents Using the NVIDIA AI Blueprint for Video Search and Summarization | NVIDIA Technical Blog
https://developer.nvidia.com/blog/advance-video-analytics-ai-agents-using-the-nvidia-ai-blueprint-for-video-search-and-summarization/

[2] [3] [67] [68] Nvidia unveils generative AI and NIM microservices for OpenUSD | Alexandre Embry
https://www.linkedin.com/posts/alexandre-embry-1a9638aa_nvidia-unveils-generative-ai-and-nim-microservices-activity-7227342070212890624-iZF6

4 5 7 8 13 14 17 18 19 22 24 33 35 36 37 40 41 47 48 55 69 Build a Video Search and Summarization Agent with NVIDIA AI Blueprint | NVIDIA Technical Blog
https://developer.nvidia.com/blog/build-a-video-search-and-summarization-agent-with-nvidia-ai-blueprint/

20 21 42 60 61 nvidia / llama-3.1-nemotron-nano-vl-8b-v1
https://docs.api.nvidia.com/nim/reference/nvidia-llama-3_1-nemotron-nano-vl-8b-v1

43 ActionRecognitionNet — Tao Toolkit - NVIDIA Docs
https://docs.nvidia.com/tao/tao-toolkit/text/cv_finetuning/pytorch/action_recognition_net.html

44 Developing and Deploying Your Custom Action Recognition ...
https://developer.nvidia.com/blog/developing-and-deploying-your-custom-action-recognition-application-without-any-ai-expertise-using-tao-and-deepstream/

50 Overview — Tao Toolkit - Computer Vision Model Zoo - NVIDIA Docs
https://docs.nvidia.com/tao/tao-toolkit/text/model_zoo/overview.html

51 52 66 NeMo Agent Toolkit | NVIDIA Developer
https://developer.nvidia.com/nemo-agent-toolkit

56 57 Simplify and Scale AI-Powered MetaHuman Deployment with ...
https://forums.developer.nvidia.com/t/simplify-and-scale-ai-powered-metahuman-deployment-with-nvidia-ace-and-unreal-engine-5/308469