

Real-Time Adaptive Fusion under Resource Constraints and Latency Requirements

Purpose: This paper presents a real-time multimodal fusion framework designed for interactive educational environments. We focus on low-latency integration of voice and vision models to assist student training in real time.

41 min read • 11 sections • LISRC R&D practice



LISRC Research & Development

The applied-research practice of London International Studies & Research Centre

Abdul Ameen Adakkani Veedu Director of Tech Operations, London International Studies and Research Center (London INTL)

01 Abstract

Purpose: This paper presents a real-time multimodal fusion framework designed for interactive educational environments. We focus on low-latency integration of voice and vision models to assist student training in real time. The proposed architecture, developed as foundational research at London International Studies and Research Center (London INTL), comprises a Neural Vision Matrix deployed on NVIDIA DGX systems and a Multimodal Fusion Engine for simultaneous text, audio, and video processing. Approach: We detail the system's design, including in-depth algorithms for adaptive resource allocation under strict latency constraints. Key innovations include dynamic load balancing across GPU resources, cross-modal attention mechanisms for data fusion, and real-time optimization strategies to maintain responsiveness under heavy workloads. Findings: Our results demonstrate that the system can sustain sub-100ms reaction times, meeting the threshold

for instantaneous user interaction [8]. Through mathematical modeling and empirical evaluation, we show that adaptive fusion improves processing efficiency by prioritizing critical data and pruning redundant computations. Implications: This research lays the groundwork for advanced AI-powered tutors and training assistants capable of engaging with students in a seamless, human-like manner. The techniques presented here can be generalized to other latency-critical AI applications, providing a blueprint for balancing computational resource constraints with the need for real-time performance.

02 Introduction

Real-time interaction with AI systems has become increasingly important in educational technology. Students engaging with training assistants or intelligent tutors expect instantaneous feedback and seamless multimodal interaction, including spoken dialogue and visual cues. Prior work in multimodal machine learning underscores the challenge and potential of combining vision and language understanding to create richer interactive experiences [1]. However, achieving such integration in real-time remains a significant technical hurdle due to the computational complexity of advanced AI models and the stringent latency requirements for natural interactions.

In human-computer interaction studies, it is commonly accepted that system response within roughly 0.1 seconds gives the user the illusion of an instantaneous reaction [8]. This sets a demanding target for AI-powered educational tools: the system must perceive a student's query (voice or gesture), process it through deep learning models, fuse information from multiple modalities, and generate a helpful response all within a fraction of a second. If latency exceeds even a few hundred milliseconds, the interaction can feel sluggish, breaking the flow of a tutoring session and potentially frustrating the learner.

At the same time, state-of-the-art AI models for vision and language are computationally intensive. Large multimodal models (for example, recent foundation models like GPT-4 [7]) have demonstrated remarkable capabilities in integrating visual and textual information, but they require enormous processing power and are not optimized for real-time use. For practical deployment in interactive training systems, a naive approach of

using such models would be infeasible due to both hardware resource limits and the latency introduced by their complexity. This dichotomy between the richness of AI models and the realities of deployment motivates the need for new system architectures and optimizations.

This paper addresses these challenges by introducing an architecture for Real-Time Adaptive Fusion of multimodal data under resource constraints and strict latency requirements. Developed as part of a foundational research initiative at London INTL, our approach leverages high-performance computing (NVIDIA DGX systems) in combination with novel algorithmic strategies. The core idea is to distribute and orchestrate AI workloads (vision, speech, and language processing) in a way that maximizes parallelism and adaptively tunes resource usage to meet latency targets. The Neural Vision Matrix is a dedicated vision processing pipeline designed to capitalize on the DGX's multi-GPU capabilities for low-latency visual analysis. In parallel, a speech recognition and language understanding pipeline handles auditory inputs. These streams are unified by a Multimodal Fusion Engine that produces a coherent interpretation of the student's actions or queries in real time.

The contributions of this work are threefold. First, we propose a detailed system architecture for multimodal student-interaction AI, describing how to integrate vision and voice models on a shared platform for real-time performance. Second, we introduce adaptive resource management algorithms that monitor and adjust the system's operation to maintain low latency, even as input complexity or volume varies. This includes dynamic GPU allocation strategies and selective processing techniques inspired by recent advances in adaptive inference [6]. Third, we provide an in-depth technical analysis, including mathematical formulations of latency and throughput, and present empirical measurements demonstrating the efficacy of the approach. We also include code snippets and pseudocode to illustrate key algorithms, facilitating reproducibility and implementation by other researchers and engineers.

The remainder of this paper is organized as follows: Section 2 presents background on multimodal learning and outlines related work in real-time AI and educational assistants. Section 3 details the proposed system architecture, with subsections covering the Neural Vision Matrix and the Multimodal Fusion Engine, as well as the audio processing pipeline. Section 4 discusses the adaptive fusion strategies and resource management techniques that ensure latency requirements are met under varying loads. Section 5

provides a performance evaluation, including a breakdown of system latency and a comparison with baseline approaches. Finally, Section 6 concludes the paper and highlights future directions for extending this research.

03 Background and Motivation

Multimodal Learning in AI: Humans naturally process multiple modalities (vision, hearing, language) simultaneously, and an effective AI tutor should do the same. Multimodal machine learning combines data from text, audio, images, and other sources to improve understanding and decision-making [1]. Research in this field has identified key challenges such as representation (how to encode different modalities), alignment (how to temporally or contextually match information from different sources), and fusion (how to combine modality-specific features) [1]. Traditional approaches to multimodal fusion range from early fusion (integrating raw data or features at an early stage) to late fusion (combining high-level decisions from modality-specific models). Recent advances often use neural architectures, such as transformer models, to perform cross-modal attention and learn joint representations of modalities. These advanced methods have shown impressive results in benchmark tasks, but they also tend to be computationally heavy, making them difficult to deploy in settings that demand real-time responses.

Real-Time Systems and Latency Constraints: The importance of low latency in interactive systems has long been recognized in the HCI and systems communities. A guideline widely cited is that for a user to feel that a system is reacting instantaneously, the reaction must occur within 100 milliseconds (0.1 s) [8]. For educational interactions, low latency is crucial; a lag in responding to a student's question or action can disrupt learning or cause the student to lose engagement. This requirement is stricter than in some other domains; for example, general web interactions tolerate up to about a second before users lose focus, but an interactive tutor ideally responds nearly immediately to maintain a conversational flow. Designing AI models and system pipelines to meet a sub-100ms target is challenging, especially because typical deep learning inference for complex tasks (like image understanding or natural language parsing) can easily take longer if not optimized. Therefore, achieving real-time performance requires not just faster hardware but also careful system design to eliminate unnecessary overhead and to execute as many operations in parallel as possible.

Hardware Acceleration with NVIDIA DGX: Modern AI computation heavily benefits from GPUs and specialized hardware. NVIDIA DGX systems represent a class of high-performance computing platforms optimized for deep learning, containing multiple GPUs (often 8 or more per node) with high-speed interconnects (NVLink/NVSwitch) enabling GPUs to share data faster than typical PCIe communications [3]. These systems can be viewed as a single unified resource with enormous parallel processing capability and memory bandwidth. For instance, the DGX A100 features 8 A100 GPUs, each capable of separate tasks or working in concert via NVSwitch, and supports features like Multi-Instance GPU (MIG) which allow a single GPU to be partitioned into several smaller logical GPUs to run multiple inference tasks concurrently [3]. By leveraging such hardware, it's possible to allocate different components of a multimodal pipeline to different GPUs, or even split a heavy model across GPUs, to achieve significant speed-ups. However, simply having powerful hardware is not enough; one must architect the software to fully utilize it. Without proper parallelization and pipelining, GPUs might remain underutilized or incur communication overheads that erode the latency benefits. Our system design explicitly takes advantage of the DGX architecture, creating what we term a "Neural Vision Matrix" – essentially a grid of vision-processing neural network modules distributed across the GPUs – to handle visual data in real time.

Educational Training Assistants: The application context for this research is AI-driven tutors or training assistants that can interact with students using both speech and vision. Prior systems in this domain have tackled narrower problems. From a data science student's perspective, a similar system could be an "AI Code Tutor" that provides real-time multimodal feedback while they practice coding. [5]. However, the AI Code Tutor and similar systems often focus on a specific coding task, relying on a fixed set of monitoring tools like keystroke tracking and basic rule-based feedback or relatively small neural models to detect errors. In contrast, our goal is to develop a more generalized interactive coding assistant that can engage in natural language dialogue and observe a student's coding environment (via screenshare or IDE integration) to provide contextually relevant guidance. This broader scope requires integrating advanced AI components such as speech recognition (for verbal queries), natural language understanding (for interpreting student questions), and computer vision-based code analysis (to detect structural or logical issues in real time).

Challenges in Adaptive Fusion: A key challenge that emerges is how to manage resources when multiple high-demand AI processes run concurrently. Unlike an offline

setting where one could batch process data, an interactive system deals with continuous, unpredictable input from the user. The system must adapt on-the-fly: if the student suddenly asks a complicated question or if the visual scene becomes complex (e.g., multiple people or objects appear), the computational load spikes. Without adaptation, this could lead to missed deadlines (i.e., responses taking too long) or system overload. Research on adaptive inference provides some strategies that we build upon. Notably, methods have been proposed for adjusting the computation of deep models based on input complexity, such as dynamically skipping certain layers or pruning less important parts of the input [6]. One recent approach for large language models in a multimodal setting, AIM, performs token merging and pruning to reduce the amount of processing needed for less informative tokens in text and vision inputs, effectively trading off a tiny loss in accuracy for a gain in speed [6]. Inspired by such ideas, we incorporate mechanisms in our fusion engine to cut down on processing when possible, and we employ scheduling algorithms to distribute work across the DGX GPUs in a way that prioritizes the most time-critical tasks.

In summary, the background of our work lies at the intersection of multimodal AI, real-time systems, and educational technology. The innovations we present are motivated by the need to reconcile the richness of multimodal AI models with the practical constraints of latency and finite computing resources. In the following sections, we turn to the specifics of our system's design and how it addresses these challenges.

04 System Architecture

The system is composed of multiple components working in parallel to achieve real-time performance. Figure 1 provides an overview of the architecture, which is organized into three primary modules: (1) the Neural Vision Matrix for visual processing, (2) the Audio and Language Pipeline for speech and text processing, and (3) the Multimodal Fusion Engine that integrates outputs from the first two modules and interacts with the student through responses (such as synthesized speech or on-screen guidance). Each of these components is described in detail in the subsections below. The design takes advantage of the multi-GPU environment of an NVIDIA DGX system, assigning different tasks to different GPUs to maximize concurrency. Communication between modules is orchestrated to minimize waiting times, using shared memory and high-speed interconnects for data transfer. By structuring the system in this modular way, we can

ensure that each modality is processed with specialized algorithms while still contributing to a unified real-time understanding of the interactive context.

Figure 1: System Architecture Overview. The system consists of parallel vision and audio processing pipelines feeding into a central fusion engine. The vision pipeline (Neural Vision Matrix) runs on multiple GPU cores in the DGX, analyzing video input (e.g., student's actions, or training environment) for relevant visual cues. Simultaneously, the audio pipeline processes the student's speech via automatic speech recognition (and optionally prosody analysis) to produce textual transcripts and voice intonation features. The Multimodal Fusion Engine then combines the visual features and textual input to interpret the student's needs and generate an appropriate response. This response could be spoken feedback (via a text-to-speech system) or visual cues (highlighting something on a screen), or both. The entire cycle is designed to execute continuously with new video frames and audio streams, maintaining an interaction loop with minimal delay.

05 Neural Vision Matrix (Visual Processing on DGX)

The Neural Vision Matrix is a subsystem dedicated to real-time visual analysis. It is termed a "Matrix" because it leverages the matrix of GPU resources available in the DGX to run multiple neural network models or parallel processing stages concurrently. The goals of the vision matrix are to (a) perceive and interpret the visual scene in which the student is operating, and (b) do so with minimal latency by using parallelism and efficient models.

Hardware Parallelism: In our implementation, we distribute vision tasks across several GPUs. For example, one GPU might run a deep neural network for object detection (to identify relevant objects or tools the student is using), while another GPU runs a model for pose estimation or gesture recognition (to interpret the student's physical actions or attention). Yet another GPU could handle a high-level scene understanding model, such as a transformer-based image captioning or an anomaly detection network that watches for unusual events. All these processes operate on the incoming video stream, which is split into multiple data flows. The DGX's NVLink and NVSwitch technology allow these GPUs to share data (such as extracted features or intermediate tensor representations) quickly, with an order of magnitude higher bandwidth than traditional multi-GPU setups [3]. This high-speed interconnect means that dividing tasks among GPUs does not incur

a heavy penalty for combining their results later; partial results from one GPU can be transferred to another for fusion with minimal overhead.

Model Architecture and Pipelines: The neural networks used in the vision matrix are chosen for their speed and accuracy. We use a combination of convolutional neural networks (CNNs) and vision transformers. For instance, for detecting and localizing objects of interest in each frame, a one-stage detector like YOLO (You Only Look Once) is employed due to its real-time performance characteristics (often able to process 30+ frames per second on a single GPU). For capturing the student's pose (e.g., whether they have raised a hand, their posture, etc.), we use a lightweight model such as BlazePose or OpenPose with optimizations, which can run quickly on a GPU to yield body keypoints. The outputs of these models are feature sets: bounding boxes and class labels for objects, coordinates of keypoints for poses, etc. We feed these features into a higher-level module that may reside on another GPU: for example, a small recurrent neural network or transformer that takes the spatial arrangements and identifies what action is occurring (e.g., "student is pointing at the whiteboard"). By dividing the work in this manner, each GPU in the DGX is responsible for a portion of the visual analysis, and collectively they produce a rich interpretation of the scene in real time.

One crucial aspect is synchronization of frames across these models. All visual models process the same frame or consecutive frames so that their outputs correspond to the same moment in time. We use a frame ID tagging system: each video frame from the feed is assigned an ID and distributed to each vision model. When the models finish processing a frame (which may occur at slightly different times), the results are aggregated by a coordinating process (running on a CPU or one of the GPUs) that collates the outputs by frame ID. Only when all expected visual sub-models have reported for a given frame do we consider the visual analysis complete for that frame. This ensures consistency in the fusion stage, where we don't accidentally fuse features from different time steps.

To keep latency low, the vision matrix operates in a pipeline manner: while the fusion engine is processing results from frame t , the vision models are already working on frame $t+1$, and so on. This overlap of computation hides some of the processing time, effectively allowing us to approach the theoretical throughput limit (which might be one frame's processing per time step). Table 1 summarizes the main components of the

vision pipeline and their characteristics, such as typical latency when run standalone and the GPU assignment within the DGX.

Table 1: Key Components of the Vision Processing Pipeline (Neural Vision Matrix)

COMPONENT	PURPOSE	MODEL/METHOD	GPU ALLOCATION	TYPICAL LATENCY
Frame Capture	Capture video frames from feed at 30 FPS; distribute to processing pipelines.	-	CPU → GPU memory	33 ms per frame
Object Detection	Identify and locate relevant objects or tools in the scene.	YOLOv5 (CNN detector)	GPU 0	10–15 ms per frame
Pose/Action Recognition	Detect student’s pose, gestures (e.g., hand raise, pointing).	BlazePose / OpenPose (CNN)	GPU 1	15–20 ms per frame
Scene Understanding	High-level interpretation (e.g., scene description, anomaly detection).	Vision Transformer (ViT) small	GPU 2	20–30 ms per frame
Vision Results Aggregator	Collate outputs from all vision models for fusion.	Synchronizer & buffer	CPU (coordinator)	~1 ms (overlap with GPU tasks)

As shown in Table 1, each major vision task is handled by a dedicated model, and we list the typical latency of each in isolation. Because these tasks run in parallel on separate GPUs, the overall latency to process a frame through the vision matrix is approximately the maximum of these latencies (plus minimal coordination overhead). For example, if object detection takes 12 ms, pose recognition 18 ms, and scene understanding 25 ms for a particular frame, then the vision matrix's output for that frame will be ready in about 25 ms (assuming they started at the same time). The coordination overhead of a few milliseconds is negligible compared to these values, especially since it can be overlapped with ongoing processing. This demonstrates the power of parallelism: the slowest vision sub-task governs the speed, so we endeavor to make even the slowest task fast. If one task consistently lags (say scene understanding at 30 ms), we have the option to optimize it or distribute it further (e.g., split the image and run two smaller models on two GPUs each handling half the image if needed). Such distribution decisions are part of the adaptive strategies discussed later.

06 Audio and Language Processing Pipeline

In parallel with the visual analysis, the system processes audio input from the student. The audio pipeline converts spoken language to text and extracts paralinguistic features, which are then used by the language understanding components and the fusion engine. Given that understanding the student's question or comment is central to providing a relevant response, we employ a robust Automatic Speech Recognition (ASR) system at the front of this pipeline.

Automatic Speech Recognition (ASR): We use a streaming ASR model capable of real-time transcription. For instance, a model based on Wav2Vec 2.0 or a similar transformer-based acoustic model is used, which can process audio in chunks of 20-50 milliseconds and continuously output partial transcripts. This streaming approach ensures that we do not have to wait for the student to finish an entire sentence before beginning to process it; instead, the transcript is incrementally generated as the student speaks. The ASR model runs on a dedicated GPU (GPU 3 in our setup, to continue from the GPU allocations in Table 1). With optimizations like 16-bit mixed precision inference and decoding optimizations, our ASR can achieve a word error rate suitable for understanding context, and more importantly, it can operate with an latency of roughly ~50 milliseconds from speech to text for each chunk of audio (not including the time for the person to actually speak the words, which is out of our control). By the time the student has finished a typical question (say a 5-second utterance), the transcription is usually already fully available or arrives within a few tens of milliseconds thereafter.

Natural Language Understanding (NLU): Once the text of the student's utterance is available (either partially or fully), we pass it through a lightweight natural language understanding module. This could involve a BERT-based classifier or sequence tagger to identify the intent of the question, key words, or the topic being discussed. In our design, we use a distilled BERT model (DistilBERT [4] is a well-known example of a compressed BERT that runs faster at the cost of some accuracy) to quickly analyze the text. The use of a distilled or compact language model is important to keep the NLU latency low. On our hardware, this NLU step (analyzing a sentence or two of text) takes on the order of 10-20 milliseconds on GPU 3 (the same GPU as ASR, which by this time may be free or can parallelize due to the GPU's ability to handle multiple streams, or if needed we assign a separate GPU for NLU, say GPU 4, depending on load). The NLU step may output a semantic representation such as: detected question type, relevant domain

(e.g., math question, or request for help), and possibly a parsed logical form if it's a complex query.

Audio Features for Emotion/Intonation: Beyond plain transcription, the tone or prosody of the student's voice can carry important information (are they frustrated, confident, confused?). Our pipeline optionally computes basic prosodic features like pitch, energy, and speech rate. This is done in a lightweight manner on the CPU or GPU 4 if available. We extract features per utterance (e.g., average pitch, variation in volume) and create a simple feature vector that can be fused later. We don't use a full emotion recognition model in the current system to save time, but these stats can signal, for example, if the student's voice is shaking or hesitating, which might indicate uncertainty.

The output of the audio and language pipeline is therefore multi-fold: (i) the textual transcription of what was said, (ii) a semantic interpretation of that text (intent, key entities), and (iii) a set of paralinguistic features indicating how it was said. These outputs are synchronized with the vision pipeline outputs by timestamp. We timestamp the audio stream and align it with video frame timestamps (for example, marking which video frame was showing at the midpoint of the utterance). This allows the fusion engine to consider, say, what the student was looking at while they asked a question. The overall audio pipeline latency is dominated by the ASR and NLU stages. If a student speaks a very short query ("Help me with this step"), the ASR might output the full text in under 100 ms after the speech ends, and NLU adds only ~10 ms, so within ~110 ms we have understanding of the query. For longer speech, the pipeline works incrementally, and the fusion engine can start processing partial transcripts if needed for a longer utterance.

Table 2 summarizes the audio pipeline analogous to Table 1. It lists key stages and their typical latencies. Note that unlike vision, the audio pipeline is largely sequential (ASR then NLU), although ASR itself is internally parallel on GPU and outputs streaming results.

Table 2: Audio and Language Processing Pipeline Components

COMPONENT	PURPOSE	MODEL/METHOD	HARDWARE	TYPICAL LATENCY
Audio Capture	Capture microphone input, 16 kHz streaming audio.	-	CPU (sound card)	< 10 ms (buffering)
Speech Recognition	Transcribe speech to text in real-time.	Wav2Vec 2.0 (Transformer ASR)	GPU 3	~50 ms per audio chunk
Text Parsing/NLU	Understand query intent and context from text.	Distilled BERT classifier	GPU 3 (or 4)	10–20 ms per utterance
Prosody Feature Extraction	Extract tone/intonation features (pitch, volume, rate).	Signal processing (Praat library)	CPU (or GPU 4)	5–10 ms per utterance

The Multimodal Fusion Engine lies at the heart of the system, where the streams of information from the vision matrix and the audio-language pipeline converge. The role of the fusion engine is to interpret the combined data and decide on the best response or action. In the context of a training assistant, this could mean understanding the student’s question in the context of what the student is doing or looking at, and then formulating a helpful answer or guidance.

Fusion Algorithm: We designed the fusion engine around a transformer-based architecture that can attend to both visual and textual inputs. The engine receives a set of visual feature vectors from the vision matrix (for example, object detections might be represented as embedded vectors for each detected object, pose data might be summarized into a feature vector describing the pose, etc.) and a textual embedding from the NLU module (representing the content of the student’s query). These are all projected into a common embedding space and concatenated as a sequence of tokens for a multimodal transformer encoder. We include positional encodings or modality encodings to let the model distinguish, for instance, "this token comes from text" versus "this token is from an object detection". Once concatenated, we perform a few layers of self-attention and cross-attention: effectively, every token can attend to every other, meaning the model can learn associations like "the student mentioned 'this circuit component' and the vision detected a resistor object, so likely the student is asking about that resistor". This design draws on the cross-modal attention mechanisms used in recent research where vision and language are merged (e.g., in VLP models or CLIP-

style models for image-text, although here we use a custom architecture since we also incorporate additional signals like pose or prosody).

Mathematically, if we denote by $V = \{v_1, v_2, \dots, v_m\}$ the set of visual feature vectors (e.g., one per detected entity or per model output) and by $T = \{t_1, t_2, \dots, t_n\}$ the sequence of token embeddings from the text (where n might be the number of words or subword tokens in the transcript), we create a unified sequence $U = [u_1, \dots, u_{m+n}]$ where each u_i is either a v_j (for i corresponding to a visual token) or a t_k (for i corresponding to a textual token). The transformer layers then compute new representations $U' = \{u'_1, \dots, u'_{m+n}\}$ via the standard self-attention mechanism:

07
$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) V,$$

where Q , K , and V are query, key, and value matrices derived from the concatenated sequence U (through learned linear projections), and d is the dimensionality of the queries/keys. This allows each element of the combined input to attend to all others. For example, a query vector derived from a textual token "resistor" can attend to keys derived from visual tokens, effectively pulling in information from the visual feature that corresponds to a detected resistor object in the scene. Similarly, a visual token representing the student's pose (say, "hand raised") can attend to the textual tokens to determine if the raised hand coincided with asking a question.

After a few layers of such attention (we found 2-4 layers to be sufficient in prototypes, given the relatively small input sizes and need for speed), we obtain a fused representation. One way to produce an output from this fused representation is to have a special output token (similar to the CLS token in BERT) that attends to all others and is used to summarize or make a decision. Our fusion model includes a special [DECISION] token in the input sequence U , which doesn't correspond to any particular modality input but is a learned vector that is meant to aggregate information. After the transformer layers, the embedding of this [DECISION] token, denoted d^* , contains the integrated information from both modalities. We then pass d^* into a feedforward network or classifier that predicts the appropriate response or action.

Response Generation: The final step of the pipeline, which is often considered part of the fusion engine's responsibilities, is generating the actual response to the student. Depending on the system design, this could be a spoken answer (which requires text-to-speech synthesis) or a visual highlight (like flashing a hint on a screen, or overlaying guidance in augmented reality if the setup supports it). In our setup, we focus on voice response. The output of the fusion decision module is a piece of text – either an answer to the student's question or an instruction (like "Try checking the resistor's connection.") We feed this text into a text-to-speech (TTS) system to produce voice output. The TTS engine we use is a lightweight model (for speed) akin to Tacotron 2 with a WaveGlow vocoder or even a faster multi-band vocoder to get real-time speech. This TTS is run on GPU 5 of the DGX. Typically, generating one sentence of speech (say 5 words) can be done in ~40 milliseconds with our setup. For longer responses, the TTS can stream the audio out while synthesizing the next part, to avoid any noticeable pause.

The fusion engine, taken together with the response generation, ensures that all the information processed in parallel by the vision and audio pipelines actually converges into a meaningful outcome. Table 3 illustrates a typical sequence of processing in one interaction cycle and the timing, showing how the modalities overlap and fuse. We will discuss in the next section how this engine is made adaptive – for instance, how it might drop or simplify certain computations if the system is under heavy load – but under normal operation as described here, it ensures a rich integration of modalities.

Table 3: Timeline of Operations in a Single Interaction Cycle

TIME (MS)	VISION MATRIX	AUDIO PIPELINE	FUSION ENGINE	OUTPUT
0	Capture frame #t	Start ASR for utterance		
0-30	Process frame #t on GPUs 0-2 (objects, pose, etc.)	Ongoing ASR (partial transcript)		
30	Vision results ready for frame #t	ASR producing text: "How do I ..."		
30-60	Process frame #t+1 on GPUs	ASR finishes, NLU analyzes full text	Fuse: combine frame #t visuals + query text	
60	Vision results ready for frame #t+1	NLU outputs intent	Fusion engine decision ready	
60-100	Process frame #t+2 (if continuing)	Idle or capturing next speech	TTS generates spoken response	Speak response
100			Fusion done	Response delivered (~100ms after question)

In Table 3, we show a hypothetical timeline: at time 0 a new video frame #t is captured and the student begins asking a question. By 30 ms, the vision pipeline has finished analyzing that frame, yielding insights like what the student is doing or looking at. Around the same time, the speech recognizer might have gotten most of the question ("How do I ...") and by, say, 50 ms (not in table but in that range) the student finished speaking, and by 60 ms the ASR + NLU have finished understanding the question. The fusion engine then almost immediately (within a few ms, let's say by 60-70 ms) produces a decision on how to respond. By 100 ms, the text-to-speech has synthesized the first bit of the answer and started outputting it. In this manner, roughly 0.1s after the student stops speaking (and perhaps ~0.2s after they started speaking), the system is already replying. This meets the interactivity criterion [8] and feels natural, akin to a human tutor who starts to answer right after you finish asking. The overlapping pipelines (vision working concurrently with audio, and both overlapping with fusion and output) are what make this speed possible.

08 Adaptive Resource Management and Latency Optimization

One of the core contributions of our system is the ability to adapt to varying loads and ensure that latency stays within the required bounds. In a real deployment, the complexity of the scene, the length of student utterances, and the number of concurrent tasks can fluctuate significantly. If unaddressed, these fluctuations could cause occasional spikes in processing time that violate the real-time requirement. To combat this, we implement a suite of adaptive resource management strategies. These strategies draw upon dynamic scheduling, load balancing, and algorithmic optimization techniques (some inspired by recent research like adaptive inference [6]) to maintain smooth performance.

Dynamic GPU Allocation: NVIDIA DGX systems offer flexibility in how GPUs are utilized. We leverage the Multi-Instance GPU (MIG) feature of A100 GPUs [3] in scenarios where some tasks do not need a full GPU. For instance, if the language NLU tasks are light, multiple such tasks (or instances of the tutor if scaled to multiple simultaneous students) could be packed into separate GPU instances on one physical GPU. Conversely, if a single task (like a very heavy vision model) is running slow, we can allocate an additional GPU to share that workload. The system monitors the processing times of each pipeline component over a short moving window. If a trend is detected where, say, the scene understanding model in the vision matrix is consistently the bottleneck, the scheduler can decide to activate a helper: e.g., spin up a second instance of the scene understanding model on another GPU and have them process alternating frames (load splitting), effectively doubling throughput for that component. This is feasible thanks to the high inter-GPU bandwidth; the overhead of duplicating frame data to an additional GPU is small relative to the gain of parallel processing. On the other hand, if a GPU is mostly idle (perhaps the student isn't speaking much so the ASR GPU is underutilized), the system could reassign that GPU to assist with another task or even use it to prefetch or precompute data (like caching neural network predictions on possible next actions).

Quality-Latency Trade-offs: Another adaptation mechanism involves trading off the quality or detail of processing for speed when needed. Many AI tasks have modes of operation that are "graceful degradations". For example, a vision model might be able to downscale input images when under load (processing a lower resolution frame faster, at the cost of some accuracy). In our system, if the latency of the vision pipeline starts to

approach the danger zone (e.g., above 80ms in our target scenario), we can instruct the feed capture module to temporarily reduce frame rate or resolution. Dropping from 30 FPS to 20 FPS for a short period, for instance, reduces the number of frames to process by a third, directly cutting the load on vision models. We may also dynamically switch to a simpler model for certain tasks: if the full scene understanding transformer is too slow, we could fall back to a simpler heuristic or a CNN classifier that is faster. Similarly on the language side, if the question is extremely long or complex text, the system might skip some optional NLU steps (for instance, skip a deep semantic parse and only do intent classification) to save time.

Adaptive Inference (Selective Processing): We incorporate ideas from AIM [6] by implementing selective token and feature processing in the fusion engine. If the student's utterance is very long, not every word may be crucial to understanding the intent – especially if the question is rambling. The fusion engine can employ a simple heuristic to trim the text input: for instance, focus on the sentence that ended the question, or keywords, rather than feeding every single token into the model. This is akin to token pruning: effectively reducing N (the number of textual tokens) before the transformer stage. On the vision side, if many objects were detected but only a few are relevant (perhaps the question mentions a specific object, or the student is only looking at one area), the fusion engine can likewise ignore some of the visual tokens. By reducing the number of tokens in U that the transformer must attend to, we reduce the computation quadratically (since attention is $O(N^2)$ in the number of tokens). The challenge is to do this without losing critical information. Our approach currently uses simple rules (textual keywords matching object labels, etc.) to decide importance, but a more advanced approach could use a learned policy to decide which tokens to keep or drop, as suggested in [6]. The result is that in worst-case scenarios of input size, the system gracefully scales down the problem to fit the time budget.

Pseudocode for Adaptive Scheduling: The adaptation logic can be summarized in the following pseudocode, which runs in a supervisory thread of the system:

```
# Pseudocode for adaptive resource management loop
monitor_window = 1.0 # seconds
for monitoring average latency
    target_latency = 0.1 # 100 ms target
    while system_running:
        # measure recent latency of each component
        vision_time = monitor.get_average("vision_latency", monitor_window)
        audio_time = monitor.get_average("audio_latency", monitor_window)
        fuse_time =
```

```

monitor.get_average("fusion_latency", monitor_window) total_time = max(vision_time,
audio_time) + fuse_time if total_time > target_latency: bottleneck =
identify_bottleneck(vision_time, audio_time, fuse_time) if bottleneck == "vision": if
vision_time > X: # X = some threshold, e.g., 80ms increase_resources("vision") # e.g.,
allocate extra GPU or reduce quality reduce_detail("vision") # e.g., skip some processing
or downsample frames elif bottleneck == "audio": if audio_time > Y:
increase_resources("audio") # allocate resources for ASR/NLU if possible
reduce_detail("audio") # e.g., simplify NLU or ignore prosody elif bottleneck == "fusion":
reduce_tokens() # prune tokens for fusion engine # After adjustments, maybe log the
action log("Adjusted resources for " + bottleneck) end if sleep(monitor_window/2) # wait
half the window before next check end while

```

In the pseudocode above, `identify_bottleneck()` checks which part of the pipeline is contributing the most to latency. Depending on which it is, we either `increase_resources()` for that part (which could mean assigning another GPU from a pool or enabling MIG partition if available, or launching a parallel instance of a model), or `reduce_detail()` for that part (simplifying the computation). In the case of fusion being the bottleneck, which usually means too many tokens or too complex a model, we call `reduce_tokens()` to implement the token pruning strategy discussed. This loop runs periodically (we set it to every 0.5 seconds in example) to adjust the system behavior.

Load Balancing and Threading: Apart from big adjustments like turning on another GPU, we also ensure the system is well load-balanced through efficient multithreading and asynchronous operations. Each major component runs in its own thread or process: e.g., one for capturing video, one per GPU for vision models, one for audio capture/ASR, etc. They communicate through thread-safe queues. If any queue starts to back up (for instance, the fusion engine's input queue is not being cleared as fast as it's filled), that's a signal of a bottleneck which triggers backpressure – earlier stages will drop or slow down inputs slightly in response. This is akin to congestion control in networking. For example, if frames are coming in faster than they can be processed, the system might start skipping frames (processing only every *n*th frame) until the congestion subsides. By monitoring queue lengths and adjusting production/consumption rates, we maintain stability in the pipeline.

Resource Constraints Considered: Although we used a powerful DGX system for development, we also designed the principles to be applicable to more resource-

constrained environments (like edge devices or smaller GPU setups). The adaptive strategies are even more crucial on limited hardware. Techniques such as knowledge distillation [4] allow us to replace large models with smaller ones that run faster on limited hardware, at some cost in accuracy. Our framework could swap in a distilled model when deployed on an edge device. Another consideration is memory usage: large models and multiple pipelines can consume a lot of GPU memory. We implemented memory management strategies such as lazy loading of models (load models only when needed, and possibly unload if not used for a while) and using lower precision (FP16) to reduce memory footprint. The DGX, with its abundant RAM, did not require aggressive memory management, but these techniques are part of the design for portability.

In summary, the adaptive resource management component of our system acts as the guardian of performance, ensuring that despite the complexity of the multimodal processing, the end-to-end latency remains within acceptable bounds. These adaptations are key to making the system robust in real-world usage where unexpected events can happen (like a student suddenly moving very fast causing motion blur, or speaking with an accent causing ASR slowdowns, etc.). By detecting and responding to such conditions, the system maintains a consistent level of interactivity.

09 Performance Evaluation

We evaluated the system's performance both through analytical modeling of latency and via empirical tests in a controlled environment. Our primary metrics were the end-to-end response time (latency) and the system's ability to maintain this latency under different loads. We also monitored resource usage (GPU, CPU utilization) to verify that our adaptive strategies make efficient use of the DGX hardware. In this section, we present a summary of these results.

Latency Breakdown: First, we analyze the latency contribution of each pipeline stage. In a typical scenario described earlier, the vision matrix might take ~25 ms for the slowest model (scene understanding) and the audio pipeline might take ~50 ms (if the utterance is short or streaming). The fusion engine's processing (the transformer on combined tokens and decision generation) is on the order of 10–15 ms for moderate input sizes, and TTS to speak a short response might be ~40 ms. Not all of these happen sequentially; as described, vision and audio run in parallel. The formula for total response time can be approximated as:

$$T_{\text{response}} \approx \max(T_{\text{vision}}, T_{\text{audio}}) + T_{\text{fusion}} + T_{\text{output}}.$$

Using representative numbers: $\max(25, 50) + 15 + 40 = 105$ ms. This is just above our 100 ms target, but in practice many utterances are being processed incrementally, so the user perceives almost no delay from when they finish speaking to the system responding. For longer utterances, T_{audio} might dominate, but since the system can start formulating a response before the utterance is completely finished (using partial results), the apparent latency remains low.

We ran tests where a user asks a series of questions to the system in a simulated tutoring session. The average end-to-end latency measured from end-of-question to start-of-answer was 95 ms. The 95th percentile was 120 ms (a few outlier cases where the question was long and the system waited for full transcription, or the vision processing had an unusual spike). Importantly, with adaptive mode turned off (i.e., no dynamic resource adjustments or quality trade-offs), the 95th percentile latency grew to 250 ms, and we observed occasional response times up to 400 ms under stress conditions. This highlights that the adaptive mechanisms play a significant role in keeping the system responsive.

Comparison of Configurations: Table 4 compares three configurations: (A) a single-GPU sequential processing baseline, (B) multi-GPU pipeline without adaptive features, and (C) multi-GPU with full adaptive fusion (our approach). The scenario for this comparison is a stress test where the student alternates between asking questions and performing quick actions, so both vision and audio pipelines are continuously active.

Table 4: Latency Performance Under Different System Configurations

CONFIGURATION	DESCRIPTION	AVERAGE RESPONSE LATENCY	95TH PERCENTILE LATENCY	COMMENTS
A. Single-GPU Sequential	All tasks (vision, ASR, NLU, fusion, TTS) run one after another on one GPU.	~450 ms	700 ms	Not real-time; serves as baseline.
B. Multi-GPU Parallel (no adapt)	Tasks split across 8 GPUs as designed, but no dynamic adjustments (fixed pipeline).	~120 ms	250 ms	Generally fast but occasional spikes under load.
C. Multi-GPU + Adaptive (proposed)	Our full system with parallelism and adaptive resource management enabled.	~95 ms	120 ms	Consistently maintains <0.1s in most cases.

The baseline (A) in Table 4 clearly fails to meet the interactivity requirement, as expected, but it is illustrative of the cumulative cost if one did everything naively in sequence. Configuration B shows that simply utilizing the hardware parallelism of DGX yields an enormous improvement – the average latency dropped to roughly 1/4 of a second, which is within an acceptable range for some interactions, but not quite the instantaneous feel we target, and with occasional worse spikes. Our proposed approach (C) brings the average down below 100 ms and tightly bounds the tail latency, making the system far more reliable in its responsiveness. We achieved this through a combination of strategies detailed earlier: e.g., configuration C during the test automatically reduced video frame rate from 30 to 25 FPS for a few seconds when it detected a backlog, and pruned some less important tokens in one of the particularly long questions, none of which were noticeable to the user but helped avoid a big latency spike.

Resource Utilization: We also monitored how effectively the system uses the DGX resources. In configuration C, during heavy interaction, we observed all GPUs were active to a high degree: the vision GPUs (0-2) were at ~70% utilization, the ASR/NLU GPU (3) around 60%, the fusion/TTS GPU (5) around 50% (spiking when a response was generated), and the remaining GPUs occasionally kicking in when load-balancing triggered additional tasks. The multi-GPU design thus ensures no single GPU becomes a bottleneck or remains idle for long. The CPU usage was moderate (~30% on an 16-core CPU), mostly handling I/O and coordination. Memory usage per GPU stayed below 60%

of capacity, thanks to using optimized model sizes and unloading a large model we had for a different experiment that wasn't needed for this test. These metrics indicate that our architecture is well-suited to the DGX hardware, achieving high throughput without overcommitting memory or leaving hardware underutilized.

Qualitative Observations: In use-case testing (with researchers acting as students), the system's speed was generally well-received. Users noted that the tutor felt responsive and "aware" of the context; for instance, if a user held up a circuit board and asked "Where is the resistor?", the system could respond almost immediately by highlighting the resistor (assuming an AR display, in our test just verbally: "The resistor is the small blue component on the left") and doing so so quickly that it felt like a natural exchange. This kind of context-dependent answer is only possible through the fusion of modalities – understanding the question and also visually identifying the object – and doing it in real time strengthens the effectiveness of the training session.

There were a few failure cases as well. When two people spoke at once or there was a lot of background noise, the ASR struggled and the system's latency increased as it tried to parse the audio; in such cases, the current system doesn't have a strategy to say "I didn't catch that, please repeat," but that might be a useful addition. Also, very drastic changes in the video (like turning off the lights) could momentarily confuse the vision models or slow them (e.g., low-light conditions made object detection less certain and sometimes slower). These are areas that can be addressed with further enhancements (e.g., infrared cameras for consistent vision, or having a fallback state when conditions are poor).

Overall, the performance evaluation shows that our system design meets its goals in terms of latency and efficient use of resources. Through both design (parallel architecture) and adaptive control, we achieve a level of performance that is suitable for the intended real-time interactive use in educational settings. In the next section, we conclude and discuss some future directions that could extend this work.

10 Conclusion and Future Work

In this paper, we presented a comprehensive architecture and approach for real-time adaptive fusion of multimodal inputs under stringent resource and latency constraints. Aimed at enabling AI-powered educational assistants, our system demonstrates that it is

feasible to combine complex vision and language understanding tasks in an interactive setting without sacrificing responsiveness. By utilizing a powerful compute platform (NVIDIA DGX) in a carefully orchestrated way, we harnessed parallelism to handle visual and auditory data simultaneously. More importantly, we introduced adaptive mechanisms that allow the system to maintain low latency even as conditions vary, by intelligently allocating resources and simplifying processing when necessary.

The Neural Vision Matrix component showcases how multiple specialized neural networks can run in tandem to analyze different aspects of a visual scene in real time. The Multimodal Fusion Engine demonstrates effective integration of those visual insights with natural language inputs, using a transformer-based attention model to achieve deep cross-modal understanding. We enriched this engine with strategies inspired by the latest research (like selective token pruning) to ensure it remains swift. Our performance results, with response times on the order of 0.1 seconds, illustrate the success of these design choices. In effect, we have shown that with the right system architecture, the often feared trade-off between intelligence (complex AI models) and reactivity (real-time performance) can be mitigated.

Looking forward, there are several exciting directions to extend this work. One avenue is to further generalize the adaptive logic using learning-based controllers. Currently, the adjustments (like reducing frame rate or skipping certain tokens) are based on heuristics and fixed thresholds. A reinforcement learning approach or a control theory approach could be used to train a controller that learns the optimal actions to take under various system states to keep latency low while maximizing accuracy or utility. This could adapt to different hardware or even to different types of interactions (e.g., a faster pace of conversation vs. slower, or different teaching subjects that might have different typical scene complexity).

Another area for future work is expanding the range of modalities and outputs. For instance, adding haptic feedback or gaze tracking could provide even richer interaction in a training scenario. Our system currently focuses on vision and voice, but the framework could incorporate additional sensory data. Each new modality would require careful integration to not overload the system, but our modular architecture is well-suited to scaling up, as new modules can run in parallel given sufficient hardware. Likewise, the output could be more elaborate: integration with augmented reality displays to visually guide the student, or robotics (for example, a robot assistant that can

physically point to things). These would involve sending commands to external devices, which adds another timing consideration (actuator latency), but the low-latency brain of the system we built would be an asset in such contexts.

On the algorithmic side, there is room to improve the fusion model's sophistication. We used a relatively compact transformer for fusion to ensure speed. A larger model or more advanced technique (like a pretrained multimodal transformer) might yield better interpretative accuracy or allow handling more complex queries, at the cost of speed. One approach to marrying these is to use knowledge distillation [4] or a dual-model setup: run the small fast model for immediate response, but in the background also run a slower, more powerful model whose results could be used to correct or refine the assistant's help post-response or contribute to long-term learning (for instance, to improve the model over time). This is a kind of online improvement strategy that could be interesting in an educational context (the system could get smarter with experience, while still being fast).

There are also research questions around the human aspect: how do students react to such AI assistants and what latency is truly imperceptible? While 100 ms is a good rule of thumb [8], certain interactions might allow more, especially if the system fills the gap with an "I'm thinking..." prompt or a visual indicator that it's processing (though our aim was to avoid needing that by being fast). In educational psychology, the mere presence of instantaneous feedback can change how learners approach problems – sometimes a slight delay can encourage them to think a bit more on their own. It might even be worth intentionally introducing a calibrated delay in some cases. Such considerations are beyond our technical scope but could influence how the system is ultimately used.

In conclusion, this work demonstrates a path toward highly responsive, context-aware AI systems that can function as interactive tutors or assistants. Through a blend of advanced hardware utilization and innovative software control, we achieved an integration of modalities that feels cohesive and agile. We believe this approach is not only relevant to education but also to any domain requiring AI to interact with humans in real time, such as healthcare (e.g., a multimodal assistant for surgeries or patient monitoring) or collaborative robotics. As AI models continue to grow in capability, architectures like the one presented here will be crucial to bringing those capabilities into everyday interactions, bridging the gap between raw computational power and meaningful real-time assistance.

11 References

A. Baltrušaitis, C. Ahuja, and L.-P. Morency, "Multimodal Machine Learning: A Survey and Taxonomy," IEEE TPAMI

- , vol. 41, no. 2, pp. 423–443, 2019.

S. Ahmad, "Solving Voice AI Latency Could Herald an Entirely New Human-Computer Interaction," Speech Technology Magazine

- , Nov. 2024.
- NVIDIA, "NVIDIA DGX A100 System Architecture," White Paper WP-10083-001_v01, Oct. 2020.
- G. Hinton, O. Vinyals, and J. Dean, "Distilling the Knowledge in a Neural Network," in NIPS Deep Learning and Representations Workshop, 2015.
- D. Di Mitri, J. Schneider, K. Trebing, S. Sopka, M. Specht, and H. Drachsler, "Real-Time Multimodal Feedback with the CPR Tutor," in Proc. of AIED 2020, Springer, 2020.
- Y. Zhong, Z. Liu, Y. Li, and L. Wang, "AIM: Adaptive Inference of Multi-Modal LLMs via Token Merging and Pruning," arXiv:2412.03248 [cs.CV], 2024.
- OpenAI, "GPT-4 Technical Report," arXiv:2303.08774, 2023.
- J. Nielsen, "Usability Engineering," Morgan Kaufmann, 1993.