LLMs ARE GENERAL ASYNCHRONOUS AGENTS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern LLMs are increasingly capable as autonomous agents, but they follow sequential interaction cycles: read, think, reply or call tools, repeat. Many real-world use cases are not sequential: voice assistants, embodied agents, and monitoring systems receive new inputs while they think or perform another task. Modern LLMs address this with specialized architectures for voice interaction and video streams, VLAs for robot control, asynchronous tool calling for API usage, and others. In this work, we generalize from different asynchronous tasks to general asynchronous agents that can adapt to different types of concurrency. To achieve this, we develop an asynchronous LLM framework that lets users (or the agents themselves) define inference coroutines with overlapping memory states. We showcase that Qwen 3.x models are capable of asynchronous operation for streaming video understanding, videogames, and monitoring, without task-specific training.

1 INTRODUCTION

Large language models (LLMs) are becoming increasingly capable autonomous agents, enabled by recent advances in reinforcement learning, tool use, and inference-time compute (Kimi Team et al., 2025a; Suzgun et al., 2023; Beeching et al., 2024). Modern LLMs can solve problems that require hours of uninterrupted reasoning, programming, and tool use (Jimenez et al., 2024; Schick et al., 2023; Gao et al., 2023). To solve these complex tasks, LLM agents follow a Thought-Action-Observation loop (Yao et al., 2023): instead of solving the problem in one go, the agent reasons and defines an action such as running code, then observes the outcome (e.g., error traceback) to inform its next step.

However, not all use cases allow for turn-based problem solving: a real-time voice assistant needs to listen while thinking and handling interruptions (Défossez et al., 2024), a self-driving car must quickly adjust to changes in traffic (Zhou et al., 2024a), and even a fully virtual monitoring agent needs to react quickly when the operating system has issues (Chen et al., 2025; Tang et al., 2026). Human "agents" do this naturally, sometimes without thinking, because we evolved to process continuous information streams and react to new stimuli (Eriksen & Schultz, 1979; Wessel & Aron, 2017).

However, artificial LLM agents are not naturally asynchronous as they were built for sequences and trained on turn-based interaction. The current state of the art treats each application with concurrency as a separate research problem. Real-time voice and video assistants are explicitly built for concurrent thinking and listening (Défossez et al., 2024; Wang et al., 2024e; Lin et al., 2025b; Huang et al., 2026a) and adjust when interrupted (OpenAI, 2024a; Cao et al., 2025). For embodied agents, Vision-Language-Action (VLA) models (Driess et al., 2023) often follow a dual actor-thinker architecture (Song et al., 2025; Tan et al., 2025) so they can react to stimuli while thinking. The latest programming harnesses let the user "steer" the agent with extra inputs during reasoning (OpenAI, 2026); others use asynchronous function calling or even multiple parallel sub-agents (Ning et al., 2024; Ginart et al., 2024; Zheng et al., 2026a). Currently, each of these research areas designs and trains agents for concurrency in their own task-specific manner.

In this work, we study whether LLM agents can be made *generally asynchronous*, similarly to how they are general tool users and few-shot learners. We hypothesize that, because modern LLMs learn to mimic human reasoning at some stage of their training, they may be able to imitate human asynchrony with proper framing. To test this, we design a general framework for defining asynchronous LLM agents without task-specific fine-tuning. Since communicating via text would be slow for many real-time applications, we leverage direct memory sharing (Rodionov et al., 2025; Zheng et al., 2026a) and extend its algorithms to support hybrid and multimodal language models. We adopt the popular async/await programming model via `asyncio` (van Rossum, 2012) to support concurrent LLM inference with asynchronous inputs and outputs, as depicted in Figure 1.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

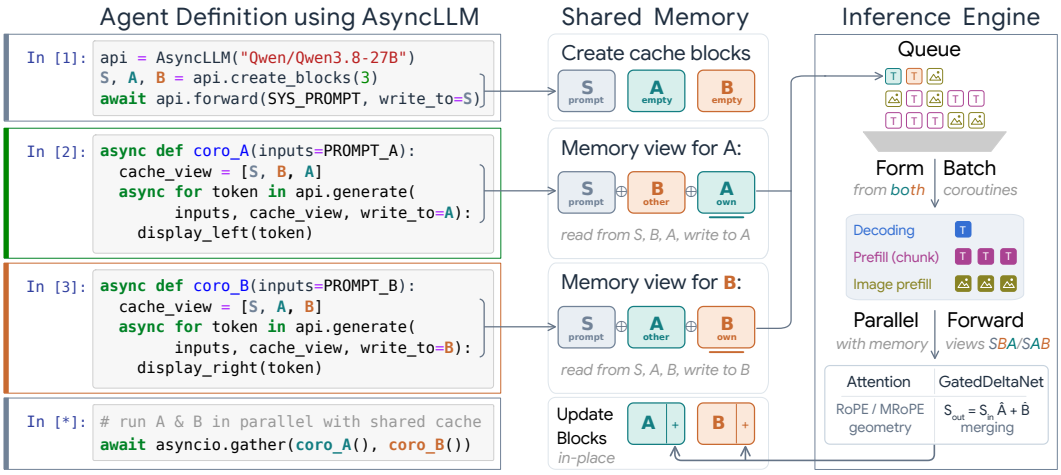


Figure 1: An example of AsyncLLM agent design with two parallel sub-agents solving a math task with shared memory: **(left)** the agent is defined as a set of coroutines that write into shared cache blocks; **(middle)** the cache blocks are arranged in "views" that define how coroutines see each other's work; **(right)** the engine groups coroutines into batches for efficient inference. Details in Section 3.

AsyncLLM lets the developer (or the agent itself) define multiple asynchronous coroutines that execute concurrently. Each coroutine writes to its own "memory block" and can access other coroutines' outputs using attention "views". This allows the developer to define coroutines that see each other's progress in real time and handle new environment I/O without waiting for current reasoning to finish. Our inference engine automatically groups concurrent inference requests for efficient batched GPU inference with shared memory, where every coroutine has its own view on the same cache blocks. Our framework lets practitioners adapt existing state-of-the-art models to new concurrency scenarios or combine multiple scenarios that would otherwise require specialized data and expensive fine-tuning. The three main contributions of this work can be summarized as follows:

- We propose AsyncLLM, a general framework for *training-free* asynchronous LLM agents in the async/await programming model. Our framework extends asynchronous programming primitives to define parallel LLM inference coroutines with overlapping memory states.
- We describe an algorithm for parallel GPU inference with shared memory states (attention KVs, GDN recurrent states), allowing multiple instances of the same LLM to run concurrently while seeing each other's progress in real time. Our algorithm supports hybrid and multimodal LLMs.[1]
- We test the generality of AsyncLLM by constructing asynchronous agents for streaming video understanding, videogame environments, and system monitoring, based on the same family of Qwen3.x VLMs without task-specific fine-tuning. Our experiments demonstrate that modern LLMs can use this programming model to define and modify their own coroutines in the same inference loop. While this capability is not yet reliable, our results suggest that future LLM generations may create self-adapting agents from environment descriptions.

2 Background

Recent works have come up with asynchronous agents across vastly different research areas. In this section, we overview several of these areas and draw parallels in how they handle concurrency.

Voice assistants (Rubenstein et al., 2023; Zhang et al., 2023b; OpenAI, 2024a; Google, 2024; Anthropic, 2025) communicate with users in real time using either an ASR-LLM-TTS pipeline or, more recently, a multimodal foundation model (Chu et al., 2023; Défossez et al., 2024; Xie & Wu, 2024; Fang et al., 2025a). However, spoken conversation requires more than multimodality (Roberts et al., 2015; Miksik et al., 2020; Mahmood et al., 2025): natural speakers ask questions while thinking, interrupt each other, and read nonverbal cues as they talk. This becomes even more pronounced in group conversation or talking while working together in a shared coding environment (Flamino et al., 2025; Houde et al., 2025; Daryanto et al., 2026; Welter et al., 2025). To maintain natural conversations, modern voice assistants work in full-duplex mode, i.e., listen, think, and speak concurrently (Défossez et al., 2024; Wang et al., 2024e; Veluri et al., 2024) with a slower background "thinker" (Lin et al., 2025b; Zhang et al., 2026; Zou et al., 2026; Wu et al., 2026b; Huang et al., 2026b). Advanced voice assistants have modules that detect interruptions ("barge-in") to pause and adjust the response (Selfridge et al., 2013; Zhao et al., 2015; Cao et al., 2025).

[1]The code is available in the anonymous supplementary materials and will be released upon publication.

In streaming video understanding (Mun et al., 2019), the model must keep up with real-time video to detect industrial incidents (Gu et al., 2024; Yang et al., 2025c; Yuan et al., 2024), assist driving (Huang et al., 2025; Zheng et al., 2026b) or comment on sporting events (Mkhallati et al., 2023; Yang et al., 2026a). As video signals are denser than audio, models typically cannot process every frame in real-time. To combat this, recent works train lightweight "probes" that determine which frames can be skipped (Wang et al., 2025c; Kim et al., 2025a; Ding et al., 2025; Yang et al., 2026a), and keep a small window of recent video frames and compress past events using text descriptions (Xu et al., 2026), hidden representations (Qian et al., 2024), or retrieval (Ning et al., 2025). Streaming video models can watch and reason concurrently to reduce response delays (Guan et al., 2026; Qian et al., 2025). The mechanisms used in these models are similar to the ones used in full-duplex voice assistants with interruption handling, but the probe is used not for voice interruptions but to detect changes in traffic situation (Fang et al., 2003; Zheng et al., 2026b), handle GUI pop-ups, or react to user's nonverbal cues (Wahlster et al., 2001; Patapati et al., 2025). Full video assistants (Liu et al., 2024b; Wang et al., 2026a; Huang et al., 2026a) and GUI computer use agents (Lin et al., 2025a; Li & Shi, 2026) use similar techniques to process multiple input streams simultaneously.

Embodied agents (Ahn et al., 2022) use Vision-Language-Action models (Zitkovich et al., 2023; Kim et al., 2025b; Sapkota et al., 2025) that process visual and text inputs and choose actions for a physical system they control. Most VLAs focus on a certain type of robotic system, such as mobile manipulators (Black et al., 2025) or humanoid robots (Bjorck et al., 2025) and require fine-tuning to adapt to a new type (Wang et al., 2025a; Sun et al., 2026), while several more recent VLAs are trained to support several different embodiments (Abeyruwan et al., 2025; Luo et al., 2026). Similar to voice and video assistants, embodied agents operate in an inherently asynchronous world and need to quickly adapt to interruptions and changes in the environment (Zhou et al., 2024a; Gonzalez-Pumariega et al., 2025; Borate et al., 2026; Cao et al., 2025). Similar to full-duplex assistants, embodied agents think and act concurrently and use dedicated subroutines for processing unexpected interruptions (Song et al., 2025; Liu et al., 2025b).

Virtual & Computer Use agents can control terminal shells (Cao et al., 2024; Singer et al., 2025; Merrill et al., 2026), web browsers (Hilton et al., 2021; Deng et al., 2023; Zheng et al., 2024a; Zhou et al., 2024b; Koh et al., 2024), virtual environments (Wang et al., 2023; Ma et al., 2024; Almeida, 2026), desktop (Wu et al., 2024b; Xie et al., 2024; Hong et al., 2024) or mobile operating systems (Zhang et al., 2023a; You et al., 2024). Their design varies between applications: a terminal agent has text-only inputs, a videogame agent requires vision and audio, and browser agents have both GUI (Koh et al., 2024; Hong et al., 2024) and text-based inputs (Hilton et al., 2021; Zhou et al., 2024b). Similarly, the need for asynchrony varies from one application to another, but follows the same general patterns. A system monitoring agent (Qi et al., 2023; Shetty et al., 2024; Tang et al., 2026) needs to process a continuous stream of logs from a running system to detect problems such as memory leaks or runaway processes, which is similar to streaming video understanding. Modern coding assistants allow users to alter an already running request via mid-turn steering (OpenAI, 2026) and ask by-the-way questions (Anthropic, 2026). Though the exact steering mechanism is not disclosed, it follows the same pattern of concurrency as voice assistant interruptions.

Parallel reasoning and tool use. Parallel to application-specific asynchronous agents, several recent lines of research use concurrency in parallel LLM reasoning (Sui et al., 2025; Ning et al., 2024; Wang et al., 2022), asynchronous function calling (Ginart et al., 2024; Gim et al., 2024; Kim et al., 2024), and recursive sub-agents (Zhu et al., 2024; Zhang & Khattab, 2025). In parallel reasoning, multiple LLM instances reason on the same problem together, solving subtasks or debating ideas (Ning et al., 2024; Jin et al., 2025). Others apply a similar technique to overlap thinking with reading a long input (Tong et al., 2025), writing a response (Yakushev et al., 2025), or waiting for tool calls (Ginart et al., 2024; Gim et al., 2024). Recent works found that giving parallel sub-instances real-time access to each other's thoughts allows them to coordinate faster (Rodionov et al., 2025; Zheng et al., 2026a) and ensure safety while reasoning (Yakushev et al., 2025; Wang et al., 2026b).

The applications we reviewed differ in modalities and deployment requirements, but they use similar patterns of concurrency. Parallel inference streams are used in both full-duplex visual assistant and for subtasks in parallel reasoning. Both streaming video understanding and system monitoring benefit from event probes. Both assistants and system monitors launch subroutines to handle interruptions. These agents use custom inference software that implements task-specific parallelism and communication and need to adapt the model for their setup. In this work, we propose a framework that generalizes between these applications without the need for task-specific training and inference.

3 ASYNCHRONOUS AGENTS

We design AsyncLLM around the async/await programming model using Python `asyncio` standard library (van Rossum, 2012), where concurrency is defined through coroutines and synchronization primitives. These coroutines run concurrent LLM inference while communicating through composable memory states (CacheBlocks) that contain attention KV caches and Gated Delta Network recurrent states (Yang et al., 2025a) for a slice of tokens. Unlike prior works, AsyncLLM groups coroutines into batched GPU execution automatically, allowing users to focus on application logic.

Figure 2 shows how these components work together for an asynchronous agent that reasons about its task and simultaneously provides the user with the running summary of its progress, similar to interleaved or asynchronous reasoning (Xie et al., 2025; Yakushev et al., 2025). The agent uses three cache blocks: one for the prompt, one for private reasoning, and one for the user-facing response. This way, the "thinker" coroutine can write new thoughts into the reasoning block while the "writer" summarizes them in the response block. The implementation consists of two coroutines: a "thinker" that produces the reasoning trace and a "writer" that summarizes it. Since the writer needs to see the current reasoning progress, its cache view (L17) contains the thinker block before its own summary, reusing the memory state within. In turn, the thinker does not need to see the writer's output to reason about the problem, so its cache view only includes the problem and its own reasoning. The thinker never explicitly communicates tokens to the writer. Instead it only notifies it about a finished paragraph, so the writer can summarize the progress by accessing the shared thinking block.

We organize the rest of this section as follows: Section 3.1 defines the AsyncLLM framework in more detail, Section 3.2 describes the algorithms for quickly reconstructing memory states from consecutive cache blocks by extending prior work on attention cache manipulation (Rodionov et al., 2025); Section 3.3 covers efficient GPU inference with attention views.

3.1 ASYNCLLM PROGRAMMING MODEL

Every forward pass in AsyncLLM writes its memory state to a `CacheBlock` that contains the model's internal memory state for a continuous chunk of tokens. For modern hybrid transformers, this corresponds to KV caches for attention layers and state transitions for Gated Delta Nets (GDN). Every forward pass writes to a single cache block, but can "see" multiple other cache blocks at once in arbitrary order. We refer to these compositions as cache views, as depicted in Figures 1 & 2. Attending to multiple cache blocks is mathematically equivalent to attending to a single conventional KV cache containing the same tokens if those blocks were encoded sequentially. If multiple coroutines update their blocks concurrently while looking at each other, the resulting memory state is not equivalent to any sequential inference, but it remains legible to the LLM as we show below. This memory-based parallelism lets asynchronous agents run parallel sub-routines that synchronize instantly. In contrast, if coroutines communicated with tokens, they would have to re-encode previously generated tokens every time the memory view changes, e.g. whenever the "thinker" in Figure 2 generates a new token.

```python
import asyncio, async_llm
api = async_llm.AsyncLLM(MODEL_NAME, **config)
prompt_block, thinker_block, writer_block = await api.create_blocks(3)
paragraph_finished = asyncio.Event()              # thinker-writer synchronization
await api.forward(PROMPT, write_to=prompt_block)  # prefill prompt into prompt_block

async def thinker_coro():
    async for token in api.generate(
        "<think>\n", cache_view=[prompt_block, thinker_block], write_to=thinker_block,
    ):  # this coroutine updates thinker_block that the writer will summarize --^
        if token == "\n\n":
            paragraph_finished.set()  # notify the writer

think_in_background = asyncio.create_task(thinker_coro())
while not think_in_background.done():
    await paragraph_finished.wait()  # wait for background thoughts
    paragraph_finished.clear()
    async for token in api.generate(
        "...</think> Summary:", cache_view=[prompt_block, thinker_block, writer_block],
        write_to=writer_block, stop_at="\n",  # writer sees --^ thoughts in real-time
    ):
        send_to_user(token)     # e.g. speak via TTS
```

Figure 2: Example AsyncLLM agent for implementing parallel reasoning and speaking (simplified).

The agent updates its cache blocks by running LLM forward passes and saving the internal state to the designated cache block. In the example above, `api.forward` (prefill) and `api.generate` use the same underlying forward pass algorithm that attends to a cache view and updates the provided cache block (`write_to`). AsyncLLM engine runs concurrent inference forward passes from different coroutines in the same batch, as we discuss in Section 3.3. Additional cache blocks can be created via `api.create_block()`, emptied with `cache_block.clear()`, and merged into one via `api.merge_blocks(A, B)`, which is equivalent to attending to both blocks but with less overhead.

Asynchronous LLM applications require the agent to react to signals: voice assistant interruptions, streaming video events, browser GUI pop-ups and, others. In our framework this can be expressed by combining LLM inference with `asyncio` synchronization primitives: events, locks, queues, and so on. For instance, consider a streaming video understanding agent that selects important frames with a "probe", then notifies a background thinking coroutine about the event. In `asyncio / async_llm`, this can be expressed with an `asyncio.Event` that notifies the background thinker of an update. When multiple event-describing coroutines compile their descriptions into a shared report, they can use an `asyncio.Lock` to ensure the output is not garbled when merging.

3.2 Inference with Multiple Cache Blocks

Next, we describe how AsyncLLM runs forward passes while attending to multiple cache blocks (`cache_view`). Recall that every cache block contains KV vectors and GDN recurrent states for a contiguous slice of tokens across all model layers. For traditional attention KV layers, we could combine KV caches by rotating every subsequent cache block's keys to their new positions according to Rotary Position Embeddings (RoPE, Su et al., 2024). However, that would require rearranging all past KVs for every forward pass, which would slow down inference. Rodionov et al. (2025) show that, for full attention with RoPE, one can compute attention without rotating previous KV blocks. Instead, they rotate only the current attention queries, keeping previous keys and values as-is. Intuitively, consider the attention dot product $\langle \rho(q, i), \rho(k, j) \rangle$, where $\rho(\cdot, \cdot)$ applies RoPE rotation that encodes the query position i or the key position j. It can be rewritten as follows:

$$\langle \rho(q, i), \rho(k, j) \rangle = \langle \rho(q, i - j), \rho(k, 0) \rangle, \text{ where } \rho(k, 0) = k \tag{1}$$

The resulting algorithm keeps past KV caches on fixed 0-based positions (0, 1, 2, ...) and computes attention by rotating the current queries relative to each block, producing equivalent attention outputs.

However, modern LLMs are not limited to traditional RoPE attention layers: most state-of-the-art open-weight models are hybrids where full attention layers are interleaved with either sliding window attention (Gemma Team et al., 2026; Abadji et al., 2026) or, more frequently, linear attention or Delta Network variants (Qwen Team, 2026b; Kimi Team et al., 2026; Zeng et al., 2026). Additionally, multimodal LLMs use multimodal rotary embeddings (MRoPE) (Wang et al., 2024d) for images, audio and video inputs. In AsyncLLM, we adopt the attention manipulation from Rodionov et al. (2025) for full attention layers and propose new algorithms for linear and multimodal attention.

Concurrent Linear Attention and Gated Delta Nets. Unlike full attention, linear attentions such as GDN and KDA use a fixed-size recurrent state $S_t \in \mathbb{R}^{d_v \times d_k}$ for each head. On every new token, the model predicts learned projections, e.g. $q_t, k_t, v_t, \alpha_t, \beta_t$ for GDNs, then updates S_t and outputs o_t:

$$S_t = S_{t-1} \left(\alpha_t \left(I - \beta_t k_t k_t^\top \right) \right) + \beta_t v_t k_t^\top, \qquad o_t = S_t q_t \tag{2}$$

When computing S_t with multiple memory blocks in `cache_view`, we reformulate the problem from per-token to per-block computation. Each CacheBlock contains a transition from the initial state before the block S_{T_0} to the state after the block $S_{T_{max}}$. For modern Delta Network variants, this transition is affine in the incoming state: linear in S_{T_0} up to an additive, input-dependent term. For convenience, let us rewrite the GDN update using auxiliary matrices A_t, B_t:

$$S_t = S_{t-1} A_t + B_t, \qquad A_t = \alpha_t \big(I - \beta_t k_t k_t^\top \big), \qquad B_t = \beta_t v_t k_t^\top. \tag{3}$$

Similarly, we can rewrite the update for S_t over several consecutive steps in terms of A_t, B_t:

$$S_t = S_{t-1} A_t + B_t = \left(S_{t-2} A_{t-1} + B_{t-1} \right) A_t + B_t = S_{t-2}\, A_{t-1} A_t + B_{t-1} A_t + B_t \tag{4}$$

$$S_t = S_{t-2}\, \hat{A}_{t,2} + \hat{B}_{t,2}, \text{ where } \hat{A}_{t,2} = A_{t-1} A_t, \quad \hat{B}_{t,2} = B_{t-1} A_t + B_t. \tag{5}$$

Or more generally, we can pack n consecutive state updates as $S_t = S_{t-n}\, \hat{A}_{t,n} + \hat{B}_{t,n}$, with

$$\hat{A}_{t,n} = \prod_{i=t-n+1}^{t} A_i, \qquad \hat{B}_{t,n} = \sum_{i=t-n+1}^{t} B_i \prod_{j=i+1}^{t} A_j, \text{ where empty products equal } I \tag{6}$$

For AsyncLLM inference with multiple cache blocks, we represent each GDN head within the block with a pair of matrices $(\hat{A}, \hat{B})$ from the above equation that summarize all steps within this block as per Eq. (6). Then, the GDN state after two consecutive cache blocks L & R can be composed as:

$$S_{LR} = (S_0\hat{A}_L + \hat{B}_L)\hat{A}_R + \hat{B}_R, \qquad (\hat{A}_L, \hat{B}_L) \circ (\hat{A}_R, \hat{B}_R) = (\hat{A}_L\hat{A}_R,\ \hat{B}_L\hat{A}_R + \hat{B}_R). \tag{7}$$

This requires $O(N_{\text{blocks}})$ instead of $O(N_{\text{tokens}})$ matrix operations per head and can be done just-in-time for a given forward pass. As coroutines progress and new tokens are added, the matrices $\hat{A}, \hat{B}$ in each block are updated incrementally per Eq. (6) with a linear transform for $\hat{A}$ and an affine transform for $\hat{B}$. If another memory view orders these blocks differently, the same matrices are multiplied in a different order. This lets AsyncLLM inference engine compose attention views with hybrid LLMs using GatedDeltaNets. If the blocks are arranged in a different order, as in Figure 1, the final memory state is computed by multiplying the same pairs of matrices in a different order. Other linear attention layers follow the same computation with slightly different definitions of the original A_t, B_t. For instance, KDA replaces the scalar decay α_t in Eq. (3) with a diagonal gate $\mathrm{Diag}(\alpha_t)$, and the rest holds verbatim. In other words, other popular linear attention variants also support memory views.

Concurrent multimodal attention. When processing image and video data, modern MLLMs use multimodal or multi-dimensional positional embeddings to encode a given image patch's row and column indices along with its position in the video or image set. MRoPE partitions key (or query) dimensions into temporal, height, and width sections and rotates each section based on a given token's frame index (temporal), grid row (height) and grid column (width) indices. If AsyncLLM attends to a cache block that contains image tokens, we rotate the current query to match the relative position in the temporal dimension and keep the two spatial axes unchanged. However, there is one more caveat that affects how cache blocks are stacked together. Since an image has multiple visual tokens for every temporal frame, a cache block with T mixed visual and text tokens spans fewer than T temporal positions, and it affects how the next block should be rotated. To account for this, we explicitly maintain per-block position spans instead of rotating by token count.

Combined with prior work on concurrent attention, this allows us run to inference on modern hybrid MLLMs with arbitrary cache views. Linear attention layers compose block-wise affine transitions, while full-attention layers use query rotation with MRoPE correction. The remaining MLP layers are invariant to cache views and can be computed normally. We discuss additional architecture variants in Appendix D. The remaining challenge is batching multiple coroutines on the same device.

3.3 Scheduling and Inference Engine

We build the AsyncLLM inference engine over mini-SGLang[2], a minimal implementation of the SGLang framework (Zheng et al., 2024b). Our reference implementation consists of three main components: **i)** the front-end that implements the `asyncio`-based AsyncLLM interface from Section 3.1, **ii)** a balanced scheduling algorithm that lets reaction coroutines complete quickly without being clogged, and **iii)** efficient batched inference kernels with memory views based on Section 3.2.

Scheduling for asynchronous agents. AsyncLLM gathers incoming forward pass requests from all coroutines into a shared queue and forms batches for parallel execution. Both prefill and generation requests are mapped to the same batched LLM forward algorithm where requests of different lentghs and cache views can run in the same batch. To avoid situations where a quick response coroutine gets "clogged" by larger background requests, our engine forms balanced batches from all active coroutines. We employ chunked prefilling (Agrawal et al., 2023), i.e. mapping long prefills into multiple mathematically equivalent forward passes that can be batched with decoding for interactivity.

Batched Inference. We take advantage of modern LLM frameworks using Paged Attention (Kwon et al., 2023; Zheng et al., 2024b), i.e. splitting the KV cache into "pages". This lets us implement CacheBlocks as a light-weight data structure that holds references to attention pages and GDN affine transitions and implements update, merge, and clear operations. When running a forward pass on a given batch, we use the existing mini-SGLang kernels for everything except inner Attention and GatedDeltaNet (after projections). We track which cache views are needed for the active batch and feed them into the corresponding kernels: full attention with query rotation and MRoPE modifications, and GDN kernels that use the algorithm in Section 3.2 to construct the GDN state, then run Flash Linear Attention (Yang & Zhang, 2024). We discuss additional implementation details in Appendix C and evaluate inference latency and throughput under different synthetic workloads in Appendix E.

[2]Based on `https://github.com/sgl-project/mini-sglang`

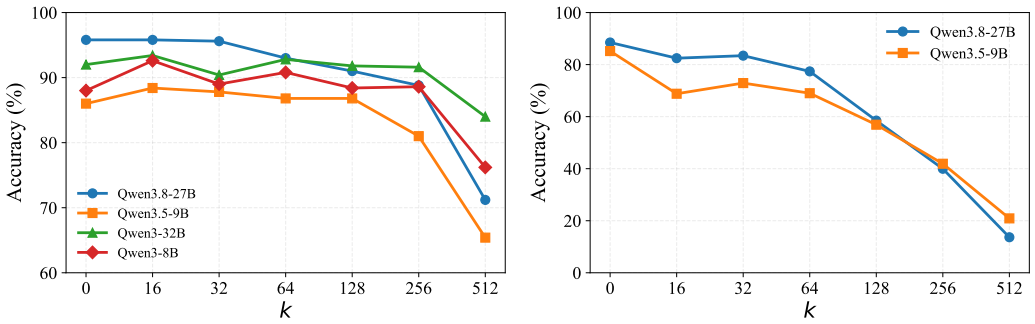


Figure 3: Accuracy with one asynchronous input; the x-axis denotes when the input arrived (step #). **(left)** text clarifications on MATH-500-Sharded, **(right)** image changes on ShardedVQA (Section 4.1).

4 Experiments

The core idea of this work is that LLM agents can be made *generally* asynchronous without training, and that memory view manipulations from Section 3.2 let them react to asynchronous inputs. To better isolate each individual claim, we begin by evaluating individual text and image changes in Section 4.1. Next, we evaluate AsyncLLM agents on real-time streaming video in Section 4.2, interactive agents for videogames in Section 4.3, and text-based system monitoring in Section 4.4.

4.1 Sanity Checks: Single Asynchronous Input

We first test the simplest asynchronous setting: mid-reasoning user clarifications. The agent begins solving a math or image-QA task, then receives a missing detail or correction while thinking. We evaluate an AsyncLLM agent equivalent to AsyncReasoning (Yakushev et al., 2025), extended to hybrid LLMs using Section 3.2, on hybrid-attention Qwen 3.5+ models (Qwen Team, 2026b).

Asynchronous text inputs. We evaluate hybrid LLMs on MATH-500-Sharded (Yakushev et al., 2025), where each of 500 math problems (Hendrycks et al., 2021) is split into an incomplete prompt and a clarification. The clarification is provided after k reasoning steps. We report these evaluations in Figure 3 (left) and provide non-hybrid AsyncReasoning for reference.

Asynchronous image changes. Next, we evaluate whether VLM-based agents can revise ongoing reasoning when visual evidence changes. We construct 513 image pairs from existing visual QA and math datasets. For each problem, we edit the input image to add realistic "errors" and use the original image as the "revised" version. The errors are constructed so that the problem cannot be solved correctly from the erroneous image. Dataset construction is described in Appendix F. The agent starts working with the erroneous (edited) image. After k decoding steps, we replace it with the corrected image and add a notification that the input has changed.

Figure 3 summarizes our results: AsyncLLM agents based on Qwen 3.x hybrid MLLMs can react to asynchronous inputs in both visual and text modalities. The visual agent shows trends similar to text interruptions, but the accuracy drops somewhat faster: upon closer inspection, we found that this is explained by the fact that visual tasks, on average, require less reasoning than MATH-500, and larger k sometimes arrive when the agent has already produced the answer. Still, our results demonstrate that AsyncLLM agents can react to inputs while reasoning. In subsequent sections, we use this ability to process visual and text streams in more complex applications.

4.2 Streaming Video Understanding

We switch from a single image change to monitoring a continuous video stream and evaluate training-free Streaming Video Understanding. We evaluate on the SoccerNet-Caption (Mkhallati et al., 2023) sports commentary dataset using the streaming evaluation protocol from Ding et al. (2025), and on the ProactiveVideoQA mixed video benchmark using the official protocol (Wang et al., 2025b).

The AsyncLLM agent consists of five concurrent components: **1) the event probe** runs every frame and determines if it contains a new event worth describing. If the probe triggers, the image pair (cache block) is passed to **2) the background thinking thread** that reconstructs video events from frames. At the same time, **3) the output probe** checks if the reasoning contains a new event. If it does, the **4) description writer** generates the event description from the thinker's internal state, and a final non-LLM **5) output compiler** coroutine gathers the results. Our agent uses *training-free probes*: instead of training a separate sub-module as in Ding et al. (2025); Yang et al. (2026a), we run the base LLM as-is with a pre-filled prompt that determines important frames in a single forward pass. We provide a detailed agent description and prompts in Appendix G.

Table 1: Streaming video understanding evaluation with AsyncLLM Qwen 3.x models and a non-AsyncLLM Mage-VL streaming model: **(left)** SoccerNet results in streaming video understanding protocol, **(right)** ProactiveVideoQA evaluation across four subsets and totals, using the metrics from the original evaluation protocol (all higher is better). See details and discussion in Section 4.2.

Agent & Model	SoccerNet (streaming)			ProactiveVideoQA PAUC (ω=0.5)					$\text{Trigger}_{\text{Acc}}$ TimVal	
	AUROC	$\text{Trigger}_{\text{Acc}}$	TimVal	WEB	EGO	TV	VAD	ALL	ALL	ALL
Qwen3.5-9B	0.677	62.82	39.62	0.493	0.563	0.638	0.358	0.541	52.99	17.19
Qwen3.8-27B	0.608	54.88	33.27	0.501	0.482	0.616	0.366	0.504	52.45	16.82
Q3.6-35B-A3B	0.651	62.55	37.46	0.476	0.578	0.624	0.344	0.539	52.30	16.37
Mage-VL	0.555	52.79	27.87	0.323	0.538	0.390	0.284	0.428	43.27	10.03

We evaluate AsyncLLM with three Qwen 3.x models and compare it against Mage-VL (Yang et al., 2026a), a recent model trained specifically for video streaming (see Appendix H for hyperparameter tuning). We follow the standard evaluation protocol for both benchmarks: event probe ROC AUC, TriggerAcc (fraction of correctly predicted events), and TimVal (which balances speak and silence correctness). On ProactiveVideoQA, we also report Proactive AUC with the recommended response delay weight (PAUC ω=0.5) and report other ω in Appendix H. Table 1 summarizes our results: AsyncLLM with Qwen3.5+ models outperforms the more specialist Mage-VL on both benchmarks. While the official evaluation protocol for SoccerNet does not measure description quality, the larger Qwen3.5+ models also provide more accurate descriptions on ProactiveVideoQA. We see this not a weakness of Mage-VL that was trained with different priorities, but a positive side-effect of using generalist MLLMs in AsyncLLM. Finally, we report inference speed in Table 3.

4.3 Interactive Agents for Video Games

Next, we move from passive monitoring to agents that affect their environment. We evaluate on two ViZDoom scenarios based on the videogame Doom (Kempka et al., 2016; Wydmuch et al., 2018): HealthGathering and DeadlyCorridor (frame skip 4). We modify the AsyncLLM agent architecture from the previous section: instead of describing the video stream, the background thinker module determines the course of action (e.g. "turn right until you are aiming at the rightmost enemy") and a faster action subroutine converts it into frame-by-frame actions with an early-exit probe that determines when the action can be picked. We compare against two baseline agents: sequential and probe-only. Before interacting, the agents are prompted with the environment rules and action space and allowed to reason about strategy. Then, on each frame, they observe two latest frames from the environment and are probed to choose action. We prompt the sequential agent to write short single-paragraph reasoning to avoid overthinking. We also evaluate a faster probe-only agent that runs one forward pass per frame with a pre-written template[3]. The results in Figure 4 show that the AsyncLLM agent can react much faster than a sequential agent based on the same model while preserving the gains from reasoning. This demonstrates that AsyncLLM can generalize existing VLMs to interactive videogame agents without training. That said, our results only demonstrate the basic capability, not state-of-the-art performance for these specific environments. We also experiment with letting **agents define their own coroutines** based on the environment and AsyncLLM API (see Appendix J), which shows good initial results on HealthGathering but inferior on DeadlyCorridor.

[3] "Based on ..., the next action is: ___", after which we take a valid action token with the highest probability.

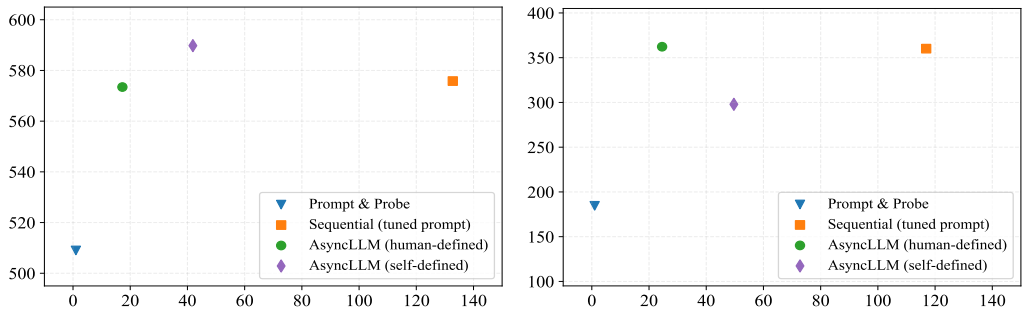


Figure 4: AsyncLLM Qwen3.6-35B-A3B evaluation on VizDoom environments **(left)** DoomHealthGathering and **(right)** DoomDeadlyCorridor; the y-axis is the average reward and the x-axis is the mean delay (forward passes) from observation to taking an action, averaged over 100 episodes.

Table 2: DevOps-Gym Monitoring results with anti-distraction prompt and streaming eval for Section 4.4.

Agent	Accuracy (%), ↑	# forward passes ↓	Mean % events ↓
Q3.6-35B-A3B	61.76	8452	100
+ early answer	47.06	6641	58.71
+ skip rows	26.47	3686	65.86
AsyncLLM	55.88	3837	55.49
Qwen3.5-9B	47.06	8726	100
+ early answer	44.12	5749	61.21
+ skip rows	8.82	2204	78.23
AsyncLLM	41.18	2582	44.98

Table 3: AsyncLLM inference speed, 1x H200 (top) decoding throughput on synthetic tasks, (bottom) real-time rate on ProactiveVideoQA.

Model	9B	27B	35B-A3B
Throughput↑, tokens per second.			
1 coros	106	43	96
2 coros	192	80	169
4 coros	339	144	284
8 coros	552	239	438
Average stream seconds↑ per GPU-second.			
Video only	4.4	2.9	2.5
+ audio (TV)	1.8	1.4	1.4

4.4 System Monitoring Agents

For our next evaluation, we analyze the AsyncLLM agent for system monitoring (Qi et al., 2023; Chen et al., 2025; Tang et al., 2026). In this setup, the agent works with an input stream of system logs from a running application and detects anomalies, such as memory leaks or runaway file descriptors. We use the Monitoring subset from the DevOps-Gym benchmark (Tang et al., 2026) consisting of 34 tasks. The original setup only measured accuracy with several types of anomalies such as network latency spikes or system handle leaks. We modify our evaluation setup to account for how quickly the system responds: we let the agents perceive logs in real-time and measure, on average, how soon the agent answered and how many inference steps it used. To make our results GPU-agnostic, we report the number of forward passes. This is equal to the number of output tokens for the sequential baseline but counts two tokens generated in the same batch as one forward pass (see in Table 3 for throughput reference). The AsyncLLM agent has, on average, 2.06 active coroutines for this task.

The AsyncLLM monitoring agent follows the same architecture as in Streaming Video Understanding except for a different input format: instead of images, the agent receives chunks of logs split by lines up to 1 second or 1024 characters long (at least one line). We compare AsyncLLM against three alternatives that use the same paradigm: a baseline sequential agent with the same model that gives the answer after the whole sequence, a sequential agent with an early-answer tool after every log chunk (same prompt as AsyncLLM), and a faster baseline agent that checks whether it should reason on a given chunk of logs or skip it entirely. We use an additional prompt paragraph that ensures the agent only worries about resource usage issues and limits its reasoning to keep up with the current events. Without that, we found that Qwen 3.6-35B-A3B was overthinking unrelated issues and fell behind real-time logs resulting in $<10\%$ accuracy. We apply the same anti-distraction prompt to AsyncLLM and baselines. Table 2 summarizes our results: for both models, the asynchronous monitoring agents can detect problems with comparable accuracy, but significantly faster than comparable sequential agents, using same concurrency principles as the streaming video agent in Section 4.2.

5 Discussion and Future Work

In this work, we investigated the ability of large language models to operate as asynchronous agents over a diverse set of environments. Our results suggest that asynchrony can be treated as a general capability rather than a collection of domain-specific skills. We found that modern MLLMs already have the capacity to handle concurrent sub-tasks and react to new stimuli without task-specific asynchronous training. Similarly to early tool-use agents, they do not always outperform purpose-trained models on their specific domains, but are much easier to adapt to new applications. To facilitate this adaptation, we formulated AsyncLLM, a framework that lets the user define asynchronous agents in the async/await programming model, and proposed efficient algorithms that extend this capability to modern hybrid and multimodal LLMs. Our experiments demonstrate that AsyncLLM agents can generalize to asynchronous text and visual updates, streaming video understanding, system monitoring, and even interactive videogame agents. This could enable real-time asynchronous APIs that let users run LLMs in asynchronous environments, e.g. by supplying the agent definition through the API. Alternatively, proprietary LLMs can write their own harness from the user's prompt.

This opens two interesting directions for future research: **i)** training LLMs to be better general asynchronous agents, in the same sense that current LLMs train to be better general tool use agents, and **ii)** further investigating the agent's ability to improve its own coroutines for the given scenario.

AI USE STATEMENT

In this work, we used generative AI tools for generating, cleaning, and reformatting a partially synthetic dataset, and minor thematic analysis during early prototyping, and we experimented with using generative AI to implement an agent in one of our experiments. In particular, we used generative AI tools for generating and cleaning synthetic incomplete images for asynchronous image inputs dataset (Appendix F). We also used a generative AI API for LLM-as-a-judge evaluation for MATH-500 (Hendrycks et al., 2021) as part of its canonical evaluation protocol. Appendix J describes the case where we used the generative AI (asynchronous agent) to write its own AsyncLLM definition for the environment it operates in. We have not used generative AI tools for developing theoretical models or conceptual frameworks, assisting with translation, proposing hypotheses, or providing feedback on research methodology. Formulating mathematical claims, providing ingredients for these claims, or assisting in writing proofs are not applicable to this work. Additionally, we used search-augmented generative AI tools to help find additional related works, and we used generative AI coding assistants to draft empty templates for certain tables and plots. We have reviewed all AI-assisted work: we checked the synthetic erroneous images manually and ran VLM inference to ensure that the problem cannot be solved without the corrected image. We also verified the Python plot templates and tables before entering our results. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work studies general-purpose asynchronous LLM agents that can observe, reason, and act concurrently. While this can improve responsiveness in applications such as streaming assistants, monitoring systems, and embodied or interactive agents, it also introduces additional risks. Concurrent execution may cause an agent to emit an incorrect or harmful action before slower reasoning or monitoring coroutines can intervene, and asynchronous inputs may change an agent's behavior while other computations are still in progress. More capable asynchronous agents may also increase the usefulness of LLMs for dual-use applications such as automated system interaction, surveillance, or other continuously operating agents. Additionally, we ran experiments with self-modifying agents that needed sandboxing and strict limits on the tools, files, and external systems that an agent is allowed to modify. Real-world deployments of AsyncLLM agents in sensitive use cases should include appropriate guardrails, sandboxing, output/action validation, and monitoring, depending on the deployment scenario.

REPRODUCIBILITY STATEMENT

To facilitate reproducibility, we provide our reference implementation of AsyncLLM and the sharded VQA dataset in the (anonymous) supplementary materials. We also provide experiment configurations and prompts used for asynchronous agent and dataset creation. The supplementary code and evaluation dataset will be released upon publication.

REFERENCES

Julien Abadji, Marah Abdin, Connor Adams, Eric Alcaide, Mustafa Altun, Michele Artoni, Junze Bao, Uday Barar, Vassilis Bekiaris, Arkadii Bessonov, Benjamin Bütikofer, Jonathan Chang, Yen-Chun Chen, Dmitry Chernenkov, Yang Chi, Filippos Christianos, Fenia Christopoulou, Razvan-Andrei Ciocoiu, Tzachi Cohen, Yohann Coppel, Dmitrii Emelianenko, Brandon Fergerson, Brian Fitzgerald, Matthias Gallé, Alex Golonzovskyi, George Grigorev, Yiyang Hao, Christian Hensel, Jan Huenermann, Ye Ji, Sarthak Joshi, Eiso Kant, Kabir Khandpur, Seonghyeon Kim, Vladimir Kirichenko, Umut Kocasarac, Ilya Kochik, Ivan Komarov, Chaerin Kong, Anurag Koul, François-Joseph Lacroix, Sergei Laktionov, Waren Long, Quentin Malartic, Vadim Markovtsev, Afonso Marques, Robert McHardy, Carlos Mocholí, Dmitry Monakhov, Adam Morris, Martin Muller, Christian Mürtz, Robin Nabel, Thien Nguyen, Rok Novosel, Szymon Ozog, Aalhad Patankar, Aleksei Petrov, Alexandre Piché, Arthur Pignet, Teodor Poncu, Phil Potter, Alexander Rakowski, Pierre-Yves Ritschard, Jay Roberts, Joe Rowell, Piotr Sarna, Pierre-André Savalle, Uladzislau Sazanovich, Nikita Shapovalov, Arsenii Shevchenko, Mikhail Shilkov, Andrei Sokol, Mohamed

Soliman, Jack Stephenson, Victor Storchan, Dragos-Constantin Tantaru, Artem Tyurin, Adrian Wälchli, Pengming Wang, Jianxiao Yang, Renat Zayashnikov, Alexander Zelenka Martin, Nikolay Zinov, Caroline Bercier, José Caldeira, Margarida Garcia, Tom George, Kabeer Gharzai, Glenn Hitchcock, Carson Klingenberg, Ivo Pinto, Varun Randery, Noah Smith, Arina Sugako, and Jason Warner. Laguna m.1/xs.2 technical report, 2026. URL https://arxiv.org/abs/2605.27605.

Saminda Abeyruwan, Joshua Ainslie, Jean-Baptiste Alayrac, Montserrat Gonzalez Arenas, Travis Armstrong, Ashwin Balakrishna, Robert Baruch, Maria Bauza, Michiel Blokzijl, Gemini Robotics Team, et al. Gemini robotics: Bringing ai into the physical world. *arXiv preprint arXiv:2503.20020*, 2025.

Amey Agrawal, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav S Gulavani, and Ramachandran Ramjee. Sarathi: Efficient llm inference by piggybacking decodes with chunked prefills. *arXiv preprint arXiv:2308.16369*, 2023.

Amey Agrawal, Nitin Kedia, Ashish Panwar, Jayashree Mohan, Nipun Kwatra, Bhargav Gulavani, Alexey Tumanov, and Ramachandran Ramjee. Taming Throughput-Latency tradeoff in LLM inference with Sarathi-Serve. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 117–134, Santa Clara, CA, July 2024. USENIX Association. ISBN 978-1-939133-40-3. URL https://www.usenix.org/conference/osdi24/presentation/agrawal.

Michael Ahn, Anthony Brohan, Noah Brown, Yevgen Chebotar, Omar Cortes, Byron David, Chelsea Finn, Chuyuan Fu, Keerthana Gopalakrishnan, Karol Hausman, Alex Herzog, Daniel Ho, Jasmine Hsu, Julian Ibarz, Brian Ichter, Alex Irpan, Eric Jang, Rosario Jauregui Ruano, Kyle Jeffrey, Sally Jesmonth, Nikhil Joshi, Ryan Julian, Dmitry Kalashnikov, Yuheng Kuang, Kuang-Huei Lee, Sergey Levine, Yao Lu, Linda Luu, Carolina Parada, Peter Pastor, Jornell Quiambao, Kanishka Rao, Jarek Rettinghouse, Diego Reyes, Pierre Sermanet, Nicolas Sievers, Clayton Tan, Alexander Toshev, Vincent Vanhoucke, Fei Xia, Ted Xiao, Peng Xu, Sichun Xu, Mengyuan Yan, and Andy Zeng. Do as i can and not as i say: Grounding language in robotic affordances. In *arXiv preprint arXiv:2204.01691*, 2022.

Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. Gqa: Training generalized multi-query transformer models from multi-head checkpoints. *arXiv preprint arXiv:2305.13245*, 2023. URL https://arxiv.org/abs/2305.13245.

Diogo Almeida. Introducing System One Models & Jev. TypeSafe AI Blog, September 2026. URL https://typesafe.ai/blog/introducing-system-one-models-and-jev.

Anthropic. Using voice mode on claude mobile apps. https://support.claude.com/en/articles/11101966-using-voice-mode-on-claude-mobile-apps, 2025. Accessed: December 1, 2025.

Anthropic. Claude code commands:
btw. https://code.claude.com/docs/en/commands, 2026. Accessed: 2026-09-20.

ARC Prize Foundation. Openai's new o3 system scores breakthrough on arc-agi-pub, 2024. URL https://arcprize.org/blog/oai-o3-pub-breakthrough. Accessed: 2025.03.28.

Zhangir Azerbayev, Hailey Schoelkopf, Keiran Paster, Marco Dos Santos, Stephen Marcus McAleer, Albert Q. Jiang, Jia Deng, Stella Biderman, and Sean Welleck. Llemma: An open language model for mathematics. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=4WnqRR915j.

Henry G Baker Jr and Carl Hewitt. The incremental garbage collection of processes. *ACM SIGART Bulletin*, pp. 55–59, 1977.

Edward Beeching, Lewis Tunstall, and Sasha Rush. Scaling test-time compute with open models, 2024. URL https://huggingface.co/spaces/HuggingFaceH4/blogpost-scaling-test-time-compute.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. Longformer: The long-document transformer, 2020. URL https://arxiv.org/abs/2004.05150.

James Betker. Better speech synthesis through scaling. *arXiv preprint arXiv:2305.07243*, 2023. Tortoise TTS: expressive multi-voice text-to-speech.

Johan Bjorck, Fernando Castañeda, Nikita Cherniadev, Xingye Da, Runyu Ding, Linxi Fan, Yu Fang, Dieter Fox, Fengyuan Hu, Spencer Huang, et al. Gr00t n1: An open foundation model for generalist humanoid robots. *arXiv preprint arXiv:2503.14734*, 2025.

Kevin Black, Noah Brown, James Darpinian, Karan Dhabalia, Danny Driess, Adnan Esmail, Michael Equi, Chelsea Finn, Niccolo Fusai, Manuel Y. Galliker, Dibya Ghosh, Lachy Groom, Karol Hausman, Brian Ichter, Szymon Jakubczak, Tim Jones, Liyiming Ke, Devin LeBlanc, Sergey Levine, Adrian Li-Bell, Mohith Mothukuri, Suraj Nair, Karl Pertsch, Allen Z. Ren, Lucy Xiaoyang Shi, Laura Smith, Jost Tobias Springenberg, Kyle Stachowicz, James Tanner, Quan Vuong, Homer Walke, Anna Walling, Haohuan Wang, Lili Yu, Ury Zhilinsky, and Physical Intelligence. $\pi_{0.5}$: a vision-language-action model with open-world generalization, 2025. URL https://arxiv.org/abs/2504.16054.

Suraj Borate, Bhavish Rai B, Vipul Pardeshi, and Madhu Vadali. Llm-based generalizable hierarchical task planning and execution for heterogeneous robot teams with event-driven replanning, 2026. URL https://arxiv.org/abs/2511.22354.

Charles Cao, Feiyi Wang, Lisa Lindley, and Zejiang Wang. Managing linux servers with llm-based ai agents: An empirical evaluation with gpt4. *Machine Learning with Applications*, 17: 100570, 2024. ISSN 2666-8270. doi: https://doi.org/10.1016/j.mlwa.2024.100570. URL https://www.sciencedirect.com/science/article/pii/S266682702400046X.

Shiye Cao, Jiwon Moon, Amama Mahmood, Victor Nikhil Antony, Ziang Xiao, Anqi Liu, and Chien-Ming Huang. Interruption handling for conversational robots. In *arXiv preprint arXiv:2501.01568*, 2025.

Shuaichen Chang, David Palzer, Jialin Li, Eric Fosler-Lussier, and Ningchuan Xiao. Mapqa: A dataset for question answering on choropleth maps. *arXiv preprint arXiv:2211.08545*, 2022.

Guangyu Chen, Yu Zhang, Jianlin Su, Weixin Xu, Siyuan Pan, Yaoyu Wang, Yucheng Wang, Guanduo Chen, Bohong Yin, Yutian Chen, Junjie Yan, Ming Wei, Y. Zhang, Fanqing Meng, Chao Hong, Xiaotong Xie, Shaowei Liu, Enzhe Lu, Yunpeng Tai, Yanru Chen, Xin Men, Haiqing Guo, Y. Charles, Haoyu Lu, Lin Sui, Jinguo Zhu, Zaida Zhou, Weiran He, Weixiao Huang, Xinran Xu, Yuzhi Wang, Guokun Lai, Yulun Du, Yuxin Wu, Zhilin Yang, and Xinyu Zhou. Attention residuals, 2026.

Joya Chen, Zhaoyang Lv, Shiwei Wu, Kevin Qinghong Lin, Chenan Song, Difei Gao, Jia-Wei Liu, Ziteng Gao, Dongxing Mao, and Mike Zheng Shou. Videollm-online: Online video large language model for streaming video. In *CVPR*, 2024.

Yinfang Chen, Manish Shetty, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Jonathan Mace, Chetan Bansal, Rujia Wang, and Saravan Rajmohan. AIOpslab: A holistic framework to evaluate AI agents for enabling autonomous clouds. In *Eighth Conference on Machine Learning and Systems*, 2025. URL https://openreview.net/forum?id=3EXBLwGxtq.

Yunfei Chu, Jin Xu, Xiaohuan Zhou, Qian Yang, Shiliang Zhang, Zhijie Yan, Chang Zhou, and Jingren Zhou. Qwen-audio: Advancing universal audio understanding via unified large-scale audio-language models. arXiv preprint arXiv:2311.07919, 2023. URL https://arxiv.org/abs/2311.07919.

Koen Claessen. A poor man's concurrency monad. *Journal of Functional Programming*, 9(3): 313–323, 1999.

Taufiq Daryanto, Xiaohan Ding, Kaike Ping, Lance T. Wilhelm, Yan Chen, Chris Brown, and Eugenia H. Rho. Human-human-ai triadic programming: Uncovering the role of ai agent and the value of human partner in collaborative learning, 2026. URL https://arxiv.org/abs/2601.12134.

K. H. Davis, R. Biddulph, and S. Balashek. Automatic recognition of spoken digits. *The Journal of the Acoustical Society of America*, 24(6):637–642, 11 1952. ISSN 0001-4966. doi: 10.1121/1.1906946. URL https://doi.org/10.1121/1.1906946.

Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Sam Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. *Advances in Neural Information Processing Systems*, 36:28091–28114, 2023.

Xin Ding, Hao Wu, Yifan Yang, Shiqi Jiang, Qianxi Zhang, Donglin Bai, Zhibo Chen, and Ting Cao. Streammind: Unlocking full frame rate streaming video dialogue through event-gated cognition. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 13448–13459. IEEE, 2025.

Danny Driess, Fei Xia, Mehdi S. M. Sajjadi, Corey Lynch, Aakanksha Chowdhery, Brian Ichter, Ayzaan Wahid, Jonathan Tompson, Quan Vuong, Tianhe Yu, Wenlong Huang, Yevgen Chebotar, Pierre Sermanet, Daniel Duckworth, Sergey Levine, Vincent Vanhoucke, Karol Hausman, Marc Toussaint, Klaus Greff, Andy Zeng, Igor Mordatch, and Pete Florence. PaLM-e: An embodied multimodal language model. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 8469–8488. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/driess23a.html.

Yilun Du, Shuang Li, Antonio Torralba, Joshua B. Tenenbaum, and Igor Mordatch. Improving factuality and reasoning in language models through multiagent debate. In *Forty-first International Conference on Machine Learning*, 2023. URL https://openreview.net/forum?id=zj7YuTE4t8.

Alexandre Défossez, Laurent Mazaré, Manu Orsini, Amélie Royer, Patrick Pérez, Hervé Jégou, Edouard Grave, and Neil Zeghidour. Moshi: a speech-text foundation model for real-time dialogue, 2024. URL https://arxiv.org/abs/2410.00037.

Vage Egiazarian, Andrei Panferov, Denis Kuznedelev, Elias Frantar, Artem Babenko, and Dan Alistarh. Extreme compression of large language models via additive quantization. *arXiv preprint arXiv:2401.06118*, 2024.

Vage Egiazarian, Roberto Castro, Denis Kuznedelev, Andrei Panferov, Eldar Kurtic, Shubhra Pandit, Alexandre Marques, Mark Kurtz, Saleh Ashkboos, Torsten Hoefler, et al. Bridging the gap between promise and performance for microscaling fp4 quantization. In *International Conference on Learning Representations*, volume 2026, pp. 113529–113563, 2026.

Charles W Eriksen and Derek W Schultz. Information processing in visual search: A continuous flow conception and experimental results. *Perception & psychophysics*, 25(4):249–263, 1979.

Chiung-Yao Fang, Sei-Wang Chen, and Chiou-Shann Fuh. Automatic change detection of driving environments in a vision-based driver assistance system. *IEEE Transactions on Neural Networks*, 14:646–657, 02 2003. doi: 10.1109/TNN.2003.811353.

Qingkai Fang, Yan Zhou, Shoutao Guo, Shaolei Zhang, and Yang Feng. LLaMA-omni 2: LLM-based real-time spoken chatbot with autoregressive streaming speech synthesis. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 18617–18629, Vienna, Austria, July 2025a. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.912. URL https://aclanthology.org/2025.acl-long.912/.

Zhen Fang, Zhuoyang Liu, Jiaming Liu, Hao Chen, Yu Zeng, Shiting Huang, Zehui Chen, Lin Chen, Shanghang Zhang, and Feng Zhao. Dualvla: Building a generalizable embodied agent via partial decoupling of reasoning and action. *arXiv preprint arXiv:2511.22134*, 2025b.

James Flamino, Mohammed Shahid Modi, Boleslaw K. Szymanski, Brendan Cross, and Colton Mikolajczyk. Testing the limits of large language models in debating humans. *Scientific Reports*, 15:13852, 2025. doi: 10.1038/s41598-025-98378-1. URL https://doi.org/10.1038/s41598-025-98378-1.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv preprint arXiv:2210.17323*, 2022.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. PAL: Program-aided language models. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 10764–10799. PMLR, 23–29 Jul 2023. URL https://proceedings.mlr.press/v202/gao23f.html.

Gemma Team, Sherif El Abd, Vaibhav Aggarwal, Robin Algayres, Alek Andreev, Olivier Bachem, Ian Ballantyne, Cormac Brick, Victor Cărbune, Michelle Casbon, Mayank Chaturvedi, Aditya Chawla, Victor Cotruta, Alice Coucke, Phil Culliton, Robert Dadashi, Lucas Dixon, Mohamed Elhawaty, Utku Evci, Clément Farabet, Johan Ferret, Filippo Galgani, Sertan Girgin, Jean-Bastien Grill, Maarten Grootendorst, Jiaxian Guo, Cassidy Hardin, Yanzhang He, Steven M. Hernandez, Omri Homburger, Léonard Hussenot, Juyeong Ji, Armand Joulin, Aishwarya Kamath, Parnian Kassraie, Olivier Lacombe, Preethi Lahoti, Gaël Liu, Gus Martins, Luciano Martins, Tatiana Matejovicova, Ramona Merhej, Nikola Momchev, Sneha Mondal, Ryan Mullins, Sindhu Raghuram Panyam, Shreya Pathak, Sarah Perrin, André Susano Pinto, Etienne Pot, Angéline Pouget, Alexandre Ramé, Sabela Ramos, Douglas Reid, David Rim, Morgane Rivière, Karsten Roth, Louis Rouillard, Omar Sanseviero, Pier Giuseppe Sessa, Shane Settle, Danila Sinopalnikov, Sara Smoot, Piotr Stanczyk, Andreas Steiner, Lawrence Stewart, Ilya Tolstikhin, Michael Tschannen, Anton Tsitsulin, Nino Vieillard, Renjie Wu, Pingmei Xu, Haichuan Yang, Edouard Yvinec, Biao Zhang, Li Zhang, Joe Zou, Nicolas Aagnes, Abdelrahman Abdelhamed, Jakub Adamek, Shivani Agrawal, Shubham Agrawal, Ibrahim Alabdulmohsin, Jean Baptiste Alayrac, Uri Alon, Chandramouli Amarnath, Ankesh Anand, Chrysovalantis Anastasiou, Setareh Ariafar, François-Xavier Aubet, Kyriakos Axiotis, Federico Barbero, Joelle Barral, Alexei Bendebury, Urs Bergmann, Stanley Bileschi, Kat Black, Mathieu Blondel, Sebastian Borgeaud, Arthur Bražinskas, Ryan Burnell, Robert Busa-Fekete, Mu Cai, Daniele Calandriello, Glenn Cameron, Charlotte Caucheteux, Rahma Chaabouni, Garima Chadha, Jetha Chan, Blake Jianhang Chen, Jesse Chen, Lin Chen, Xu Chen, Derek Cheng, Tzu hsiang Chien, Nikolai Chinaev, Yi Chou, Zhaohui Chu, Benjamin Coleman, Pooja Consul, Sam Conway-Rahman, Scott Crowell, Dylan Cutler, Vivek Dani, Samira Daruki, Anil Das, Daniel Deutsch, Nishanth Dikkala, Li Ding, Qiuhan Ding, Shenil Dodhia, Konstantin Donhauser, Tulsee Doshi, Anca Dragan, Alex Druinsky, Sahil Dua, Zoltan Egyed, Danielle Eisenbud, Daniel Eppens, Cindy Fan, Bahare Fatemi, Yassir Fathullah, Vlad Feinberg, Milen Ferev, Sebastian Flennerhag, Takumi Fujimoto, João Gabriel Oliveira, Isaac Galatzer-Levy, João Gante, Simon Geisler, Soham Ghosal, Antonious M. Girgis, Tamara von Glehn, Alec Go, Alhaad Gokhale, Alex Grills, Yiming Gu, Mayank Gupta, Pramod Gupta, Guru Guruganesh, Raia Hadsell, Hamza Harkous, Jitendra Harlalka, Demis Hassabis, Anja Hauth, Joe Heyward, Arian Hosseini, Chih-Yang Hsia, I-Hung Hsu, Xiaopeng Huang, Yangsibo Huang, Kevin Hui, Adrian Hutter, Te I, Fotis Iliopoulos, Advait Jain, Ganesh Jawahar, Ziwei Ji, Qilin Jin, Melvin Johnson, Kandarp Joshi, Arun Kandoor, Wang-Cheng Kang, Koray Kavukcuoglu, Mehran Kazemi, Kathleen Kenealy, Amr Khalifa, Phoebe Kirk, Ivan Korotkov, Suraj Kothawade, Vitaly Kovalev, Neel Kovelamudi, Adam Kraft, Ravin Kumar, Vivek Kumar, Harish Kuppam, Justin Lannin, Chen-Yu Lee, Seungji Lee, Dmitry Lepikhin, Alon Levkovitch, Dongdong Li, Qiujia Li, Valentin Liévin, Ethan Lin, Ziqian Lin, Casper Liu, Tianlin Liu, Tianqi Liu, Xin Liu, Ivan Lobov, Mayank Lunayach, Min Ma, Gagan Madan, Andrii Maksai, Eric Malmi, Michal Matuszak, Daniel McDuff, Gaurav Menghani, Maciej Mikuła, Daniil Mirylenka, Karolis Misiunas, Vedant Misra, Andreea Mitran, Kareem Mohamed, Maksim Mukha, Eric Noland, James O'Donnell, Brendan O'Donoghue, Kate Olszewska, Bernett Orlando, Wanqiong Pan, Rina Panigrahy, Unnati Parekh, Nicolas Perez-Nieves, Chunjong Park, Eric Paskie, Liqian Peng, Bryce Petrini, Slav Petrov, Jonas Pfeiffer, Bilal Piot, Martyna Plomecka, Siim Poder, Octavio Ponce, Arijit Pramanik, David Racz, Anish Rajan, Michelle Ramanovich, Anand Rao, Marvin Ritter, Vitor Rodrigues, Evan Rosen, Mikołaj Rybiński, Noveen Sachdeva, Michaël E. Sander, Rohit Sathyanarayana, Sagar Savla, Samuel Schmidgall, Tal Schuster, George Scrivener, Benoit Seguin, Andrew Sellergren, Aliaksei Severyn, Izhak Shafran, Dhruv Shah, Bobak Shahriari, Yuan Shangguan, Ashish Shenoy, Pradeep Shenoy, Rakesh Shivanna, Pauline Sho, Lucas Spangher, Wojciech Stokowiec, Tim Strother, Yao Su, Yinghao Sun, Mukund Sundararajan, Andrea Tacchetti, Mor Hazan Taege, Pouya Tafti, Jean Tarbouriech, Chetan Tekur, Shantanu Thakoor, Rahul Thapa, Madeleine Traverse, Lenart Treven, Tao Tu, Chien Te Tung, Çağlar Ünlü, Petar Veličković, Malini Pooni Venkat, Sagar Gubbi Venkatesh, Vidya Venkiteswaran, Francesco Visin, Alex Vitvitskyi, Kiran Vodrahalli, Weiyi Wang, Xin Wang, Tris Warkentin, Jan Wassenberg, John Wieting, Cindy Wu, Lechao Xiao, Hao Xu, Yuhui Xu, Fuzhao Xue, Arun Yadav, Jun Yan,

Antoine Yang, Lin Yang, Ming-Hsuan Yang, Ziyu Ying, Jae Hyeon Yoo, Morteza Zadimoghaddam, Sajjad Zafar, Fred Zhang, Jiageng Zhang, Jianyi Zhang, Xiaofan Zhang, Chao Zhao, David Zhou, and Chen Zou. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

In Gim, Seung seob Lee, and Lin Zhong. Asynchronous llm function calling, 2024. URL https://arxiv.org/abs/2412.07017.

Antonio A Ginart, Naveen Kodali, Jason Lee, Caiming Xiong, Silvio Savarese, and John Emmons. Asynchronous tool usage for real-time agents. *arXiv preprint arXiv:2410.21620*, 2024.

Gonzalo Gonzalez-Pumariega, Leong Su Yean, Neha Sunkara, and Sanjiban Choudhury. Robotouille: An asynchronous planning benchmark for llm agents, 2025. URL https://arxiv.org/abs/2502.05227.

Google. Gemini live (voice mode). https://gemini.google/overview/gemini-live/, 2024. Accessed: 2025-12-01.

Google. Gemini live api overview. https://ai.google.dev/gemini-api/docs/live-api, September 2026. Accessed: 2026-09-24.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Zhaopeng Gu, Bingke Zhu, Guibo Zhu, Yingying Chen, Ming Tang, and Jinqiao Wang. Anomalygpt: detecting industrial anomalies using large vision-language models. In *Proceedings of the Thirty-Eighth AAAI Conference on Artificial Intelligence and Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence and Fourteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'24/IAAI'24/EAAI'24. AAAI Press, 2024. ISBN 978-1-57735-887-9. doi: 10.1609/aaai.v38i3.27963. URL https://doi.org/10.1609/aaai.v38i3.27963.

Yiran Guan, Liang Yin, Dingkang Liang, Jianzhong Ju, Zhenbo Luo, Jian Luan, Yuliang Liu, and Xiang Bai. Video streaming thinking: Videollms can watch and think simultaneously. *arXiv preprint arXiv:2603.12262*, 2026.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.

Jacob Hilton, R Nakano, S Balaji, and John Schulman. Webgpt: Improving the factual accuracy of language models through web browsing. *OpenAI Blog, December*, 16, 2021.

Wenyi Hong, Weihan Wang, Qingsong Lv, Jiazheng Xu, Wenmeng Yu, Junhui Ji, Yan Wang, Zihan Wang, Yuxiao Dong, Ming Ding, et al. Cogagent: A visual language model for gui agents. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14281–14290. IEEE, 2024.

Coleman Hooper, Sehoon Kim, Hiva Mohammadzadeh, Michael W Mahoney, Yakun S Shao, Kurt Keutzer, and Amir Gholami. Kvquant: Towards 10 million context length llm inference with kv cache quantization. *Advances in Neural Information Processing Systems*, 37:1270–1303, 2024.

Stephanie Houde, Kristina Brimijoin, Michael Muller, Steven I. Ross, Dario Andres Silva Moran, Gabriel Enrique Gonzalez, Siya Kunde, Morgan A. Foreman, and Justin D. Weisz. Controlling ai agent participation in group conversations: A human-centered approach. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, IUI '25, pp. 390–408, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400713064. doi: 10.1145/3708359.3712089. URL https://doi.org/10.1145/3708359.3712089.

Xueyu Hu, Tao Xiong, Biao Yi, Zishu Wei, Ruixuan Xiao, Yurun Chen, Jiasheng Ye, Meiling Tao, Xiangxin Zhou, Ziyu Zhao, et al. Os agents: A survey on mllm-based agents for general computing devices use. *arXiv preprint arXiv:2508.04482*, 2025.

Lianghua Huang, Zhi-Fan Wu, Wei Wang, Yupeng Shi, Mengyang Feng, Junjie He, Chen-Wei Xie, Yu Liu, Jingren Zhou, Ang Wang, et al. Wan-streamer v0. 1: End-to-end real-time interactive foundation models. *arXiv preprint arXiv:2606.25041*, 2026a.

Muye Huang, Lingling Zhang, Xingyu Yu, Lei Shi, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Jun Liu. Duplexomni: Real-time listening, seeing, thinking, and speaking for full-duplex interaction, 2026b. URL https://arxiv.org/abs/2606.09186.

Zhenpeng Huang, Xinhao Li, Jiaqi Li, Jing Wang, Xiangyu Zeng, Cheng Liang, Tao Wu, Xi Chen, Liang Li, and Limin Wang. Online video understanding: Ovbench and videochat-online. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3328–3338. IEEE, 2025.

Saurabh Jha, Rohan Arora, Yuji Watanabe, Takumi Yanagawa, Yinfang Chen, Jackson Clark, Bhavya Bhavya, Mudit Verma, Harshit Kumar, Hirokuni Kitahara, et al. Itbench: Evaluating ai agents across diverse real-world it automation tasks. *arXiv preprint arXiv:2502.05352*, 2025.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. Mistral 7b. *arXiv preprint arXiv:2310.06825*, 2023a.

Yunfan Jiang, Agrim Gupta, Zichen Zhang, Guanzhi Wang, Yongqiang Dou, Yanjun Chen, Li Fei-Fei, Anima Anandkumar, Yuke Zhu, and Linxi Fan. VIMA: Robot manipulation with multimodal prompts. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 14975–15022. PMLR, 23–29 Jul 2023b. URL https://proceedings.mlr.press/v202/jiang23b.html.

Carlos E Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. Swe-bench: Can language models resolve real-world github issues? In *International Conference on Learning Representations*, volume 2024, pp. 54107–54157, 2024.

Tian Jin, Ellie Y. Cheng, Zack Ankner, Nikunj Saunshi, Blake M. Elias, Amir Yazdanbakhsh, Jonathan Ragan-Kelley, Suvinay Subramanian, and Michael Carbin. Learning to keep a promise: Scaling language model decoding parallelism with learned asynchronous decoding, 2025. URL https://arxiv.org/abs/2502.11517.

Justin Johnson, Bharath Hariharan, Laurens Van Der Maaten, Li Fei-Fei, C Lawrence Zitnick, and Ross Girshick. Clevr: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2901–2910, 2017.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are RNNs: Fast autoregressive transformers with linear attention. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 5156–5165. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/katharopoulos20a.html.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers. *Advances in Neural Information Processing Systems*, 36:24892–24928, 2023.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. ViZDoom: A Doom-based AI research platform for visual reinforcement learning. In *IEEE Conference on Computational Intelligence and Games*, pp. 341–348, Santorini, Greece, Sep 2016. IEEE. URL http://arxiv.org/abs/1605.02097. The best paper award.

Junhyeok Kim, Min Soo Kim, Jiwan Chung, Jungbin Cho, Jisoo Kim, Sungwoong Kim, Gyeongbo Sim, and Youngjae Yu. Egospeak: Learning when to speak for egocentric conversational agents in the wild, 2025a. URL https://arxiv.org/abs/2502.14892.

Moo Jin Kim, Karl Pertsch, Siddharth Karamcheti, Ted Xiao, Ashwin Balakrishna, Suraj Nair, Rafael Rafailov, Ethan P Foster, Pannag R Sanketi, Quan Vuong, Thomas Kollar, Benjamin Burchfiel, Russ Tedrake, Dorsa Sadigh, Sergey Levine, Percy Liang, and Chelsea Finn. Openvla: An open-source vision-language-action model. In Pulkit Agrawal, Oliver Kroemer, and Wolfram Burgard (eds.), *Proceedings of The 8th Conference on Robot Learning*, volume 270 of *Proceedings of Machine Learning Research*, pp. 2679–2713. PMLR, 06–09 Nov 2025b. URL https://proceedings.mlr.press/v270/kim25c.html.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. An llm compiler for parallel function calling. In *Forty-first International Conference on Machine Learning*, 2024.

Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, Zhuofu Chen, Jialei Cui, Hao Ding, Mengnan Dong, Angang Du, Chenzhuang Du, Dikang Du, Yulun Du, Yu Fan, Yichen Feng, Kelin Fu, Bofei Gao, Hongcheng Gao, Peizhong Gao, Tong Gao, Xinran Gu, Longyu Guan, Haiqing Guo, Jianhang Guo, Hao Hu, Xiaoru Hao, Tianhong He, Weiran He, Wenyang He, Chao Hong, Yangyang Hu, Zhenxing Hu, Weixiao Huang, Zhiqi Huang, Zihao Huang, Tao Jiang, Zhejun Jiang, Xinyi Jin, Yongsheng Kang, Guokun Lai, Cheng Li, Fang Li, Haoyang Li, Ming Li, Wentao Li, Yanhao Li, Yiwei Li, Zhaowei Li, Zheming Li, Hongzhan Lin, Xiaohan Lin, Zongyu Lin, Chengyin Liu, Chenyu Liu, Hongzhang Liu, Jingyuan Liu, Junqi Liu, Liang Liu, Shaowei Liu, T. Y. Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yibo Liu, Yiping Liu, Yue Liu, Zhengying Liu, Enzhe Lu, Lijun Lu, Shengling Ma, Xinyu Ma, Yingwei Ma, Shaoguang Mao, Jie Mei, Xin Men, Yibo Miao, Siyuan Pan, Yebo Peng, Ruoyu Qin, Bowen Qu, Zeyu Shang, Lidong Shi, Shengyuan Shi, Feifan Song, Jianlin Su, Zhengyuan Su, Xinjie Sun, Flood Sung, Heyi Tang, Jiawen Tao, Qifeng Teng, Chensi Wang, Dinglu Wang, Feng Wang, Haiming Wang, Jianzhou Wang, Jiaxing Wang, Jinhong Wang, Shengjie Wang, Shuyi Wang, Yao Wang, Yejie Wang, Yiqin Wang, Yuxin Wang, Yuzhi Wang, Zhaoji Wang, Zhengtao Wang, Zhexu Wang, Chu Wei, Qianqian Wei, Wenhao Wu, Xingzhe Wu, Yuxin Wu, Chenjun Xiao, Xiaotong Xie, Weimin Xiong, Boyu Xu, Jing Xu, Jinjing Xu, L. H. Xu, Lin Xu, Suting Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ziyao Xu, Junjie Yan, Yuzi Yan, Xiaofei Yang, Ying Yang, Zhen Yang, Zhilin Yang, Zonghan Yang, Haotian Yao, Xingcheng Yao, Wenjie Ye, Zhuorui Ye, Bohong Yin, Longhui Yu, Enming Yuan, Hongbang Yuan, Mengjie Yuan, Haobing Zhan, Dehao Zhang, Hao Zhang, Wanlu Zhang, Xiaobin Zhang, Yangkun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Haotian Zhao, Yikai Zhao, Huabin Zheng, Shaojie Zheng, Jianren Zhou, Xinyu Zhou, Zaida Zhou, Zhen Zhu, Weiyu Zhuang, and Xinxing Zu. Kimi k2: Open agentic intelligence, 2025a. URL https://arxiv.org/abs/2507.20534.

Kimi Team, Yu Zhang, Zongyu Lin, Xingcheng Yao, Jiaxi Hu, Fanqing Meng, Chengyin Liu, Xin Men, Songlin Yang, Zhiyuan Li, Wentao Li, Enzhe Lu, Weizhou Liu, Yanru Chen, Weixin Xu, Longhui Yu, Yejie Wang, Yu Fan, Longguang Zhong, Enming Yuan, Dehao Zhang, Yizhi Zhang, T. Y. Liu, Haiming Wang, Shengjun Fang, Weiran He, Shaowei Liu, Yiwei Li, Jianlin Su, Jiezhong Qiu, Bo Pang, Junjie Yan, Zhejun Jiang, Weixiao Huang, Bohong Yin, Jiacheng You, Chu Wei, Zhengtao Wang, Chao Hong, Yutian Chen, Guanduo Chen, Yucheng Wang, Huabin Zheng, Feng Wang, Yibo Liu, Mengnan Dong, Zheng Zhang, Siyuan Pan, Wenhao Wu, Yuhao Wu, Longyu Guan, Jiawen Tao, Guohong Fu, Xinran Xu, Yuzhi Wang, Guokun Lai, Yuxin Wu, Xinyu Zhou, Zhilin Yang, and Yulun Du. Kimi linear: An expressive, efficient attention architecture, 2025b. URL https://arxiv.org/abs/2510.26692.

Kimi Team, Tongtong Bai, Yifan Bai, Yiping Bao, M. C., Jianfeng Cai, Xinyuan Cai, Peizhou Cao, Yuxuan Cao, Ziwei Chai, Y. Charles, H. S. Che, Guanduo Chen, Guangyu Chen, Guanzheng Chen, Huarong Chen, Jia Chen, Jianlong Chen, Jun Chen, Kexin Chen, Peng Chen, Ruijue Chen, Wentao Chen, Xin Chen, Yang Chen, Yanru Chen, Yifei Chen, Yingjiang Chen, Yuankun Chen, Yujie Chen, Yutian Chen, Zhirong Chen, Dazhi Cheng, Yean Cheng, Jialei Cui, Jingbing Cui, Anqi Dai, Jiaqi Deng, Hao Ding, Rui Ding, Shaofeng Ding, Mengfan Dong, Mengnan Dong, Yuhao Dong, Yuxin Dong, Angang Du, Chenzhuang Du, Dikang Du, Jusen Du, Yulun Du, Yu Fan, Jing Feng, Qiulin Feng, Yichen Feng, Kelin Fu, Qiang Fu, Fuxuan Gao, Hongcheng Gao, Jingyue Gao, Tong Gao, Weijia Gao, Shangyi Geng, Jie Gong, Linhu Gong, Shengao Gong, Xiaochen Gong, Qizheng Gu, Yicheng Gu, Shuhao Guan, Haiqing Guo, Shiqi Guo, Xiang Guo, Zhengyan Guo, Beixi Hao, Wenxin Hao, Xiaoru Hao, Dailan He, Haotian He, Lehan He, Qi He, Weiran He, Xinran He, Xinyi

He, Yibo He, Yunjia He, Chao Hong, Tiange Hong, Hao Hu, Jiaxi Hu, Ruikun Hu, Weiming Hu, Yangyang Hu, Zhenxing Hu, Liang Hua, Jinbin Huang, Ke Huang, Ruiyuan Huang, Siying Huang, Weixiao Huang, Yan Huang, Zhengjie Huang, Zhiqi Huang, Yulong Hui, Chaobo Jia, Yutong Jiang, Zhejun Jiang, Zuoyou Jiang, Wenyi Jin, Xinyi Jin, Yu Jing, Huanjun Kong, Guokun Lai, Aidi Li, Cheng Li, Chengyuan Li, Cong Li, Fang Li, Guanyu Li, Haoyang Li, Jia Li, Junxiong Li, Lei Li, Letian Li, Lincan Li, Weihong Li, Wentao Li, Xintong Li, Yang Li, Yishen Li, Yiwei Li, Yuxiao Li, Zhaowei Li, Zhaoxi Li, Zheming Li, Zhengxiao Li, Zhiyuan Li, Jiawei Lin, Xiaohan Lin, Yibo Lin, Zichao Lin, Ziyan Lin, Bill Liu, Boxiao Liu, Chuan Liu, Liang Liu, Shaowei Liu, Shudong Liu, Shuran Liu, Tianwei Liu, Weizhou Liu, Yangyang Liu, Yanming Liu, Yibo Liu, Yipeng Liu, Zhengying Liu, Zhiheng Liu, Enzhe Lu, Haoyu Lu, Linqiang Lu, Tingzhan Lu, Zhiyuan Lu, Aotian Luo, G. Luo, Junyu Luo, Yifan Luo, B. Lyu, Wenzhou Lyu, Shaoguang Mao, Yuan Mei, Xin Men, Minqing Ni, Yixuan Niu, Siyuan Pan, Shujun Peng, Zhangyang Qi, Ruoyu Qin, ZeChao Qin, Zeyu Qin, Haiquan Qiu, Jianxin Qiu, Jiezhong Qiu, Bowen Qu, Yuhao Qu, Zeyu Shang, Youbo Shao, Han Shen, Jincheng Shi, Juanfeng Shi, Lidong Shi, Shengyuan Shi, Wingchun Siu, Pengwei Song, Xiaoxi Song, Jianlin Su, Yunfeng Su, Zhaochen Su, Lin Sui, Jingsong Sun, Junyao Sun, Shaoning Sun, Shuzhe Sun, Tongyu Sun, Yujun Sun, Yunpeng Tai, Chuning Tang, Heyi Tang, Sirui Tang, Zecheng Tang, Chaoran Tian, Rongpeng Tian, Yu Tian, Wei Tu, Chensi Wang, Chuang Wang, Chunjie Wang, Dinglu Wang, Feng Wang, Hailong Wang, Haiming Wang, Hao Wang, Hao Wang, Huaqing Wang, Hui Wang, Jiayi Wang, Jinglong Wang, Jinhong Wang, Jiuzheng Wang, Linian Wang, Shaobo Wang, Shenzhi Wang, Shuyi Wang, Si Wang, Siyuan Wang, Tianfu Wang, Wenjue Wang, Xingran Wang, Xinmei Wang, Xinyuan Wang, Xusheng Wang, Yalin Wang, Yangkun Wang, Yao Wang, Yaoyu Wang, Yejie Wang, Yiqin Wang, Yucheng Wang, Yuzhi Wang, Zhaoji Wang, Zhaowei Wang, Zhengtao Wang, Zhenhao Wang, Zhongsheng Wang, Zifan Wang, Chu Wei, Ming Wei, Shouxin Wei, Zichen Wen, Fan Wu, Haoning Wu, Rucong Wu, Wenhao Wu, Xiaoxue Wu, Yingcong Wu, Yongqi Wu, Yuxin Wu, Zijian Wu, Xinglang Xian, Chenxuan Xiang, Yuye Xiang, Bocheng Xiao, Chenjun Xiao, Xin Xiao, Jin Xie, Xiaotong Xie, Yifeng Xie, Zhe Xie, Bowei Xing, Yiming Xiong, Baosheng Xu, Boyu Xu, Jiale Xu, Jianfan Xu, Jing Xu, Jinjing Xu, L. H. Xu, Qingtao Xu, Shuyao Xu, Suting Xu, Tiantian Xu, Tianxiang Xu, Weixin Xu, Xinran Xu, Yangchuan Xu, Ye Xu, Yueni Xu, Ziyao Xu, Haonan Xue, Junjie Yan, Yaoyao Yan, Fan Yang, Guangyao Yang, Hao Yang, Junwei Yang, Ruoyu Yang, Wenjie Yang, Xiaofei Yang, Xinyu Yang, Yi Yang, Yiling Yang, Ying Yang, Yuchen Yang, Zhen Yang, Zhilin Yang, Zian Yang, Zuhao Yang, Haotian Yao, Dan Ye, Haoran Ye, Wenjie Ye, Zhanbo Ye, Bohong Yin, Haoxiang Yin, Xietong Yin, Chengzhen Yu, Haozhen Yu, Longhui Yu, Shengnan Yu, Shuying Yu, Tianxiang Yu, Enming Yuan, Mengjie Yuan, Tongtian Yue, Wei Yue, Yang Yue, Dunyuan Zha, Haobing Zhan, B. H. Zhang, Dehao Zhang, Fei Zhang, Hao Zhang, Haoyuan Zhang, Huanyu Zhang, Jiapei Zhang, Jiaxuan Zhang, Jin Zhang, Kaiyi Zhang, Miaozhen Zhang, Puqi Zhang, Qinglei Zhang, Rong Zhang, Rui Zhang, Shaoshuai Zhang, Shiyi Zhang, Xiaobin Zhang, Xiaoyun Zhang, Y. Zhang, Yangkun Zhang, Ye Zhang, Yichi Zhang, Yikun Zhang, Yizhi Zhang, Yongting Zhang, Yu Zhang, Yutao Zhang, Yutong Zhang, Zheng Zhang, Zijing Zhang, Bin Zhao, Chenguang Zhao, Feifan Zhao, Jinglun Zhao, Jinxiang Zhao, Shuai Zhao, Wenshuo Zhao, Xiangyu Zhao, Xuanle Zhao, Yikai Zhao, Zijia Zhao, Haozhi Zheng, Huabin Zheng, Ruihan Zheng, Shaojie Zheng, Tengyang Zheng, Haofeng Zhong, Lei Zhong, Longguang Zhong, M. Zhou, Qiankang Zhou, Runjie Zhou, Ruozhang Zhou, Xinyu Zhou, Yiqiao Zhou, Zaida Zhou, Jinguo Zhu, Liya Zhu, Xinhao Zhu, Yangjunfeng Zhu, Yuxuan Zhu, Zhen Zhu, Chen Zhuang, Weiyu Zhuang, and Xinxing Zu. Kimi k3: Open frontier intelligence, 2026. URL https://arxiv.org/abs/2607.24653.

Jing Yu Koh, Robert Lo, Lawrence Jang, Vikram Duvvur, Ming Lim, Po-Yu Huang, Graham Neubig, Shuyan Zhou, Russ Salakhutdinov, and Daniel Fried. Visualwebarena: Evaluating multimodal agents on realistic visual web tasks. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 881–905, 2024.

Jungil Kong, Jaehyeon Kim, and Jaekyoung Bae. Hifi-gan: generative adversarial networks for efficient and high fidelity speech synthesis. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA, 2020. Curran Associates Inc. ISBN 9781713829546.

Thomas Kwa, Ben West, Joel Becker, Amy Deng, Katharyn Garcia, Max Hasin, Sami Jawhar, Megan Kinniment, Nate Rush, Sydney Von Arx, et al. Measuring ai ability to complete long tasks. *arXiv preprint arXiv:2503.14499*, 2025.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626, 2023.

Wonbeom Lee, Jungi Lee, Junghwan Seo, and Jaewoong Sim. Infinigen: Efficient generative inference of large language models with dynamic {KV} cache management. In *18th USENIX symposium on operating systems design and implementation (OSDI 24)*, pp. 155–172, 2024.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. Fast inference from transformers via speculative decoding. In *International Conference on Machine Learning*, pp. 19274–19286. PMLR, 2023.

Bojie Li and Noah Shi. Agent-computer observation interfaces enable dynamic computer use, 2026. URL https://arxiv.org/abs/2606.29472.

Chengshu Li, Jacky Liang, Andy Zeng, Xinyun Chen, Karol Hausman, Dorsa Sadigh, Sergey Levine, Li Fei-Fei, Fei Xia, and Brian Ichter. Chain of code: Reasoning with a language model-augmented code emulator. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28259–28277. PMLR, 2024a. URL https://proceedings.mlr.press/v235/li24ar.html. arXiv preprint arXiv:2312.04474.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. Eagle: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, pp. 31147–31162. PMLR, 2024b.

Kevin Qinghong Lin, Linjie Li, Difei Gao, Zhengyuan Yang, Shiwei Wu, Zechen Bai, Weixian Lei, Lijuan Wang, and Mike Zheng Shou. Showui: One vision-language-action model for gui visual agent. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2025a.

Yueqian Lin, Zhengmian Hu, Jayakumar Subramanian, Qinsi Wang, Nikos Vlassis, Hai "Helen" Li, and Yiran Chen. Asyncvoice agent: Real-time explanation for llm planning and reasoning, 2025b. URL https://arxiv.org/abs/2510.16156.

Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024a.

Aixin Liu, Aoxue Mei, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, et al. Deepseek-v3. 2: Pushing the frontier of open large language models. *arXiv preprint arXiv:2512.02556*, 2025a.

Jihao Liu, Zhiding Yu, Shiyi Lan, Shihao Wang, Rongyao Fang, Jan Kautz, Hongsheng Li, and Jose M Alvare. Streamchat: Chatting with streaming video. *arXiv preprint arXiv:2412.08646*, 2024b.

Xiaoyu Liu, Chaoyou Fu, Chi Yan, Chu Wu, Haihan Gao, Yi-Fan Zhang, Shaoqi Dong, Cheng Qian, Bin Luo, Xiuyong Yang, et al. Vita-e: Natural embodied interaction with concurrent seeing, hearing, speaking, and acting. *arXiv preprint arXiv:2510.21817*, 2025b.

Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. *arXiv preprint arXiv:2402.02750*, 2024c.

Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. *arXiv preprint arXiv:2209.14610*, 2022.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. Mathvista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*, volume 2024, pp. 23439–23554, 2024.

Hao Luo, Ye Wang, Wanpeng Zhang, Sipeng Zheng, Ziheng Xi, Chaoyi Xu, Haiweng Xu, Haoqi Yuan, Chi Zhang, Yiqing Wang, et al. Being-h0. 5: Scaling human-centric robot learning for cross-embodiment generalization. *arXiv preprint arXiv:2601.12993*, 2026.

Weiyu Ma, Qirui Mi, Yongcheng Zeng, Xue Yan, Yuqiao Wu, Runji Lin, Haifeng Zhang, and Jun Wang. Large language models play starcraft ii: Benchmarks and a chain of summarization approach. *Advances in neural information processing systems*, 37:133386–133442, 2024.

Amama Mahmood, Junxiang Wang, Bingsheng Yao, Dakuo Wang, and Chien-Ming Huang. User interaction patterns and breakdowns in conversing with llm-powered voice assistants. *International Journal of Human-Computer Studies*, 195:103406, 2025. ISSN 1071-5819. doi: https://doi.org/10.1016/j.ijhcs.2024.103406. URL https://www.sciencedirect.com/science/article/pii/S1071581924001897.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. ChartQA: A benchmark for question answering about charts with visual and logical reasoning. In Smaranda Muresan, Preslav Nakov, and Aline Villavicencio (eds.), *Findings of the Association for Computational Linguistics: ACL 2022*, pp. 2263–2279, Dublin, Ireland, May 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.findings-acl.177. URL https://aclanthology.org/2022.findings-acl.177/.

Mike Merrill, Alexander Shaw, Nicholas Carlini, Boxuan Li, Harsh Raj, Ivan Bercovich, Lin Shi, Jeong Shin, Thomas Walshe, E Kelly Buchanan, et al. Terminal-bench: Benchmarking agents on hard, realistic tasks in command line interfaces. In *International Conference on Learning Representations*, volume 2026, pp. 40903–40986, 2026.

Ondrej Miksik, I. Munasinghe, J. Asensio-Cubero, S. Reddy Bethi, S.-T. Huang, S. Zylfo, Xuechen Liu, T. Nica, A. Mitrocsak, S. Mezza, Rory Beard, Ruibo Shi, Raymond W. M. Ng, Pedro A. M. Mediano, Zafeirios Fountas, S.-H. Lee, J. Medvesek, Hongbin Zhuang, Yvonne Rogers, and Pawel Swietojanski. Building proactive voice assistants: When and how (not) to interact. *CoRR*, abs/2005.01322, 2020. URL https://arxiv.org/abs/2005.01322.

Hassan Mkhallati, Anthony Cioppa, Silvio Giancola, Bernard Ghanem, and Marc Van Droogenbroeck. Soccernet-caption: Dense video captioning for soccer broadcasts commentaries. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 5074–5085. IEEE, 2023.

Ruaridh Mon-Williams, Gen Li, Ran Long, Wenqian Du, Christopher G. Lucas, et al. Embodied large language models enable robots to complete complex tasks in unpredictable environments. *Nature Machine Intelligence*, 7:592–601, 2025. doi: 10.1038/s42256-025-01005-x.

Jonghwan Mun, Linjie Yang, Zhou Ren, Ning Xu, and Bohyung Han. Streamlined dense video captioning, 2019. URL https://arxiv.org/abs/1904.03870.

John Nagle. On packet switches with infinite storage. *IEEE transactions on communications*, 35(4): 435–438, 1987.

Xuefei Ning, Zinan Lin, Zixuan Zhou, Zifu Wang, Huazhong Yang, and Yu Wang. Skeleton-of-thought: Prompting LLMs for efficient parallel generation. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=mqVgBbNCm9.

Zhenyu Ning, Guangda Liu, Qihao Jin, Chengwei Li, Wenchao Ding, Minyi Guo, and Jieru Zhao. Livevlm: Efficient online video understanding via streaming-oriented kv cache and retrieval. *arXiv preprint arXiv:2505.15269*, 2025.

OpenAI. Hello gpt-4o. Online technical report, 2024a. URL https://openai.com/index/hello-gpt-4o. Voice-mode multimodal model supporting audio, text, and vision. Available at https://openai.com/index/hello-gpt-4o.

OpenAI. Introducing the realtime api. https://openai.com/index/introducing-the-realtime-api/, October 2024b. Accessed: 2026-09-24.

OpenAI. Mid-turn steering. https://platform.openai.com/api/docs/guides/steering, 2026. OpenAI API documentation.

Santosh Patapati, Aashrith Tatineni, and Trisanth Srinivasan. Geneca: A general-purpose framework for real-time adaptive multimodal embodied conversational agents. In *Proc. Interspeech 2025*, pp. 3541–3542, 2025.

Daniel Povey, Arnab Ghoshal, Gilles Boulianne, Lukas Burget, Ondrej Glembek, Nagendra Goel, Mirko Hannemann, Petr Motlicek, Yanmin Qian, Petr Schwarz, Jan Silovsky, Georg Stemmer, and Karel Vesely. The kaldi speech recognition toolkit. `https://kaldi-asr.org`, 2011. Open-source speech recognition toolkit.

Ryan Prenger, Rafael Valle, and Bryan Catanzaro. Waveglow: A flow-based generative network for speech synthesis. In *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3617–3621, 2019. doi: 10.1109/ICASSP.2019.8683143.

Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Shu Yang, Carol Fung, Hailong Yang, Depei Qian, Jing Shang, Zhiwen Xiao, and Zhihui Wu. Loggpt: Exploring chatgpt for log-based anomaly detection. In *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 273–280. IEEE, 2023.

Rui Qian, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Shuangrui Ding, Dahua Lin, and Jiaqi Wang. Streaming long video understanding with large language models, 2024. URL `https://arxiv.org/abs/2405.16009`.

Rui Qian, Shuangrui Ding, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Yuhang Cao, Dahua Lin, and Jiaqi Wang. Dispider: Enabling video llms with active real-time interaction via disentangled perception, decision, and reaction, 2025. URL `https://arxiv.org/abs/2501.03218`.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. URL `https://arxiv.org/abs/2307.16789`.

Qwen Team. On the design of Qwen3.8-Next architecture: Evaluation, efficiency, and training stability. Technical report, Alibaba Group, August 2026a.

Qwen Team. Qwen3.5: Towards native multimodal agents, February 2026b. URL `https://qwen.ai/blog?id=qwen3.5`.

Qwen Team. Qwen3.8-max: A new bar for coding and cowork, August 2026c. URL `https://qwen.ai/blog?id=qwen3.8`.

Alec Radford, Jong Wook Kim, Tao Xu, Greg Brockman, Christine Mcleavey, and Ilya Sutskever. Robust speech recognition via large-scale weak supervision. In Andreas Krause, Emma Brunskill, Kyunghyun Cho, Barbara Engelhardt, Sivan Sabato, and Jonathan Scarlett (eds.), *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 28492–28518. PMLR, 23–29 Jul 2023. URL `https://proceedings.mlr.press/v202/radford23a.html`.

Seán G. Roberts, Francisco Torreira, and Stephen C. Levinson. The effects of processing and sequence organization on the timing of turn taking: a corpus study. *Frontiers in Psychology*, Volume 6 - 2015, 2015. ISSN 1664-1078. doi: 10.3389/fpsyg.2015.00509. URL `https://www.frontiersin.org/journals/psychology/articles/10.3389/fpsyg.2015.00509`.

Gleb Rodionov, Roman Garipov, Alina Shutova, George Yakushev, Erik Schultheis, Vage Egiazarian, Anton Sinitsin, Denis Kuznedelev, and Dan Alistarh. Hogwild! inference: Parallel llm generation via concurrent attention, 2025. URL `https://arxiv.org/abs/2504.06261`.

Paul K. Rubenstein, Chulayuth Asawaroengchai, Duc Dung Nguyen, Ankur Bapna, Zalán Borsos, Félix de Chaumont Quitry, Peter Chen, Dalia El Badawy, Wei Han, Eugene Kharitonov, Hannah Muckenhirn, Dirk Padfield, James Qin, Danny Rozenberg, Tara Sainath, Johan Schalkwyk, Matt Sharifi, Michelle Tadmor Ramanovich, Marco Tagliasacchi, Alexandru Tudor, Mihajlo Velimirović, Damien Vincent, Jiahui Yu, Yongqiang Wang, Vicky Zayats, Neil Zeghidour, Yu Zhang, Zhishuai Zhang, Lukas Zilka, and Christian Frank. Audiopalm: A large language model that can speak and listen. arXiv preprint arXiv:2306.12925, 2023. URL https://arxiv.org/abs/2306.12925.

Pascal J Sager, Benjamin Meyer, Peng Yan, Rebekka von Wartburg-Kottler, Layan Etaiwi, Aref Enayati, Gabriel Nobel, Ahmed Abdulkadir, Benjamin F Grewe, and Thilo Stadelmann. A comprehensive survey of agents for computer use: Foundations, challenges, and future directions. *Journal of Artificial Intelligence Research*, 85, 2026.

Ranjan Sapkota, Yang Cao, Konstantinos I Roumeliotis, and Manoj Karkee. Vision-language-action models: Concepts, progress, applications and challenges. *arXiv e-prints*, pp. arXiv–2505, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/d842425e4bf79ba039352da0f658a906-Paper-Conference.pdf.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers. In *International conference on machine learning*, pp. 9355–9366. PMLR, 2021.

Steffen Schneider, Alexei Baevski, Ronan Collobert, and Michael Auli. wav2vec: Unsupervised pre-training for speech recognition. In *Interspeech 2019*, pp. 3465–3469, 2019. doi: 10.21437/Interspeech.2019-1873.

Ethan Selfridge, Iker Arizmendi, Peter Heeman, and Jason Williams. Continuously predicting and processing barge-in during a live spoken dialogue task. In Maxine Eskenazi, Michael Strube, Barbara Di Eugenio, and Jason D. Williams (eds.), *Proceedings of the SIGDIAL 2013 Conference*, pp. 384–393, Metz, France, August 2013. Association for Computational Linguistics. URL https://aclanthology.org/W13-4063/.

Noam Shazeer. Fast transformer decoding: One write-head is all you need. *arXiv preprint arXiv:1911.02150*, 2019.

Jonathan Shen, Ruoming Pang, Ron J. Weiss, Mike Schuster, Navdeep Jaitly, Zongheng Yang, Zhifeng Chen, Yu Zhang, Yuxuan Wang, Rj Skerrv-Ryan, Rif A. Saurous, Yannis Agiomvrgiannakis, and Yonghui Wu. Natural tts synthesis by conditioning wavenet on mel spectrogram predictions. In *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 4779–4783, 2018. doi: 10.1109/ICASSP.2018.8461368.

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yue Ting Zhuang. Hugginggpt: Solving ai tasks with chatgpt and its friends in hugging face. *ArXiv*, abs/2303.17580, 2023. URL https://api.semanticscholar.org/CorpusID:257833781.

Ying Sheng, Shiyi Cao, Dacheng Li, Banghua Zhu, Zhuohan Li, Danyang Zhuo, Joseph E Gonzalez, and Ion Stoica. Fairness in serving large language models. In *18th USENIX Symposium on Operating Systems Design and Implementation (OSDI 24)*, pp. 965–988, 2024.

Manish Shetty, Yinfang Chen, Gagan Somashekar, Minghua Ma, Yogesh Simmhan, Xuchao Zhang, Jonathan Mace, Dax Vandevoorde, Pedro Las-Casas, Shachee Mishra Gupta, Suman Nath, Chetan Bansal, and Saravan Rajmohan. Building ai agents for autonomous clouds: Challenges and design principles. In *Proceedings of 15th ACM Symposium on Cloud Computing*, 2024.

Brian Singer, Keane Lucas, Lakshmi Adiga, Meghna Jain, Lujo Bauer, and Vyas Sekar. On the feasibility of using llms to execute multistage network attacks, 01 2025.

Haoming Song, Delin Qu, Yuanqi Yao, Qizhi Chen, Qi Lv, Yiwen Tang, Modi Shi, Guanghui Ren, Maoqing Yao, Bin Zhao, Dong Wang, and Xuelong Li. Hume: Introducing system-2 thinking in visual-language-action model, 2025. URL https://arxiv.org/abs/2505.21432.

Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063, 2024. ISSN 0925-2312. doi: https://doi.org/10.1016/j.neucom.2023.127063. URL https://www.sciencedirect.com/science/article/pii/S0925231223011864.

Yang Sui, Yu-Neng Chuang, Guanchu Wang, Jiamu Zhang, Tianyi Zhang, Jiayi Yuan, Hongyi Liu, Andrew Wen, Shaochen Zhong, Na Zou, Hanjie Chen, and Xia Hu. A survey on efficient reasoning for large language models. *arXiv preprint arXiv:2503.16419*, mar 2025. URL https://arxiv.org/abs/2503.16419. Version 4 (last updated August 21, 2025).

Jingwen Sun, Wenyao Zhang, Zekun Qi, Shaojie Ren, Zezhi Liu, Hanxin Zhu, Guangzhong Sun, Xin Jin, and Zhibo Chen. Vla-jepa: Enhancing vision-language-action model with latent world model. In *European Conference on Computer Vision*, pp. 478–497. Springer, 2026.

Mirac Suzgun, Nathan Scales, Nathanael Scharli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V. Le, Ed H. Chi, Denny Zhou, and Jason Wei. Challenging big-bench tasks and whether chain-of-thought can solve them. In *Findings of the Association for Computational Linguistics: ACL 2023*, pp. 13003–13051, Toronto, Canada, 2023. URL https://api.semanticscholar.org/CorpusID:252917648.

Don Syme, Tomas Petricek, and Dmitry Lomov. The f# asynchronous programming model. In *International Symposium on Practical Aspects of Declarative Languages*, pp. 175–189. Springer, 2011.

Xudong Tan, Yaoxin Yang, Peng Ye, Jialin Zheng, Bizhe Bai, Xinyi Wang, Jia Hao, and Tao Chen. Think twice, act once: Token-aware compression and action reuse for efficient inference in vision-language-action models. *arXiv preprint arXiv:2505.21200*, 2025.

Yuheng Tang, Kaijie Zhu, Bonan Ruan, Chuqi Zhang, Michael Yang, Hongwei Li, Suyue Guo, Tianneng Shi, Zekun Li, Christopher Kruegel, et al. Devops-gym: Benchmarking ai agents in software devops cycle. In *International Conference on Learning Representations*, volume 2026, pp. 13021–13045, 2026.

Junlong Tong, Yingqi Fan, Anhao Zhao, Yunpu Ma, and Xiaoyu Shen. Streamingthinker: Large language models can think while reading, 2025. URL https://arxiv.org/abs/2510.17238.

Mark Towers, Ariel Kwiatkowski, John Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Kallinteris Andreas, Markus Krimmel, Arjun Kg, Rodrigo Perez-Vicente, et al. Gymnasium: A standard interface for reinforcement learning environments. *Advances in Neural Information Processing Systems*, 38, 2026.

Noriko Umeda, E Matsui, Torazo Suzuki, and Hiroshi Omura. Synthesis of fairy tales using an analog vocal tract. In *Proceedings of 6th International Congress on Acoustics*, pp. B159–162, 1968.

Aaron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. In *Proceedings of the 9th ISCA Speech Synthesis Workshop*, 2016. URL https://arxiv.org/abs/1609.03499.

Guido van Rossum. Pep 3156: Asynchronous io support rebooted: The asyncio module. Python Enhancement Proposal 3156, Python Software Foundation, December 2012. URL https://peps.python.org/pep-3156/. Accessed 2026-09-02.

Bandhav Veluri, Benjamin N Peloquin, Bokai Yu, Hongyu Gong, and Shyamnath Gollakota. Beyond turn-based interfaces: Synchronous LLMs as full-duplex dialogue agents. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 21390–21402, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-main.1192. URL https://aclanthology.org/2024.emnlp-main.1192/.

Wolfgang Wahlster, Norbert Reithinger, and Anselm Blocher. Smartkom: Towards multimodal dialogues with anthropomorphic interface agents. In Gottfried Wolf and Gunther Klein (eds.), *Proceedings of the International Status Conference "Human-Computer Interaction". International*

Status Conference Human-Computer Interaction, Germany, pp. 23–34, Berlin, Germany, 10 2001. DLR.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *Transactions on Machine Learning Research (TMLR)*, 2024a. doi: 10.48550/arXiv.2305.16291.

Haibo Wang, Bo Feng, Zhengfeng Lai, Mingze Xu, Shiyu Li, Weifeng Ge, Afshin Dehghan, Meng Cao, and Ping Huang. Streambridge: Turning your offline video large language model into a proactive streaming assistant. *Advances in Neural Information Processing Systems*, 38:132332–132359, 2026a.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. *Advances in Neural Information Processing Systems*, 37:95095–95169, 2024b.

Ke Wang, Houxing Ren, Aojun Zhou, Zimu Lu, Sichun Luo, Weikang Shi, Renrui Zhang, Linqi Song, Mingjie Zhan, and Hongsheng Li. Mathcoder: Seamless code integration in llms for enhanced mathematical reasoning. In *International Conference on Learning Representations*, volume 2024, pp. 5009–5042, 2024c.

Peng Wang, Shuai Bai, Sinan Tan, Shijie Wang, Zhihao Fan, Jinze Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, et al. Qwen2-vl: Enhancing vision-language model's perception of the world at any resolution. *arXiv preprint arXiv:2409.12191*, 2024d.

Peng Wang, Songshuo Lu, Yaohua Tang, Sijie Yan, Wei Xia, and Yuanjun Xiong. A full-duplex speech dialogue scheme based on large language model. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024e. URL https://openreview.net/forum?id=YawXY6mWiK.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed H. Chi, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *ArXiv*, abs/2203.11171, 2022. URL https://api.semanticscholar.org/CorpusID:247595263.

Xunguang Wang, Yuguang Zhou, Qingyue Wang, Zongjie Li, Ruixuan Huang, Zhenlan Ji, Pingchuan Ma, and Shuai Wang. Beyond content safety: Real-time monitoring for reasoning vulnerabilities in large language models. *arXiv preprint arXiv:2603.25412*, 2026b.

Yihao Wang, Pengxiang Ding, Lingxiao Li, Can Cui, Zirui Ge, Xinyang Tong, Wenxuan Song, Han Zhao, Wei Zhao, Pengxu Hou, Siteng Huang, Yifan Tang, Wenhui Wang, Ru Zhang, Jianyi Liu, and Donglin Wang. Vla-adapter: An effective paradigm for tiny-scale vision-language-action model, 2025a. URL https://arxiv.org/abs/2509.09372.

Yueqian Wang, Xiaojun Meng, Yifan Wang, Huishuai Zhang, and Dongyan Zhao. Proactivevideoqa: A comprehensive benchmark evaluating proactive interactions in video large language models. *arXiv preprint arXiv:2507.09313*, 2025b.

Yueqian Wang, Xiaojun Meng, Yuxuan Wang, Jianxin Liang, Jiansheng Wei, Huishuai Zhang, and Dongyan Zhao. VideoLLM knows when to speak: Enhancing time-sensitive video comprehension with video-text duet interaction format. In Christos Christodoulopoulos, Tanmoy Chakraborty, Carolyn Rose, and Violet Peng (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2025*, pp. 6338–6359, Suzhou, China, November 2025c. Association for Computational Linguistics. ISBN 979-8-89176-335-7. doi: 10.18653/v1/2025.findings-emnlp.336. URL https://aclanthology.org/2025.findings-emnlp.336/.

Yuxuan Wang, R. J. Skerry-Ryan, Daisy Stanton, Yonghui Wu, Ron J. Weiss, Navdeep Jaitly, Zongheng Yang, Ying Xiao, Zhifeng Chen, Samy Bengio, Quoc Le, Yannis Agiomyrgiannakis, Rob Clark, and Rif A. Saurous. Tacotron: Towards end-to-end speech synthesis. In *INTERSPEECH*, pp. 4006–4010. ISCA, 2017. doi: 10.21437/Interspeech.2017-1452. URL https://arxiv.org/abs/1703.10135.

Zihao Wang, Shaofei Cai, Guanzhou Chen, Anji Liu, Xiaojian (Shawn) Ma, and Yitao Liang. Describe, explain, plan and select: Interactive planning with llms enables open-world multitask agents. In A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (eds.), *Advances in Neural Information Processing Systems*, volume 36, pp. 34153–34189. Curran Associates, Inc., 2023. URL https://proceedings.neurips.cc/paper_files/paper/2023/file/6b8dfb8c0c12e6fafc6c256cb08a5ca7-Paper-Conference.pdf.

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, et al. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *Advances in Neural Information Processing Systems*, 37:113569–113697, 2024f.

Yifei Wei, Linqing Zhong, Yi Liu, Yuxiang Lu, Xindong He, Maoqing Yao, and Guanghui Ren. Libravla: Achieving learning equilibrium via asynchronous coarse-to-fine dual-system. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 39799–39815, 2026.

Alisa Welter, Niklas Schneider, Tobias Dick, Kallistos Weis, Christof Tinnes, Marvin Wyrich, and Sven Apel. From developer pairs to ai copilots: A comparative study on knowledge transfer, 2025. URL https://arxiv.org/abs/2506.04785.

Hao Wen, Yifan Su, Feifei Zhang, Yunxin Liu, Yunhao Liu, Ya-Qin Zhang, and Yuanchun Li. Parathinker: Native parallel thinking as a new paradigm to scale llm test-time compute. *arXiv preprint arXiv:2509.04475*, 2025.

Jan R Wessel and Adam R Aron. On the globality of motor suppression: unexpected events and their influence on behavior and cognition. *Neuron*, 93(2):259–280, 2017.

Bingyang Wu, Yinmin Zhong, Zili Zhang, Shengyu Liu, Fangyue Liu, Yuanhang Sun, Gang Huang, Xuanzhe Liu, and Xin Jin. Fast distributed inference serving for large language models, 2024a. URL https://arxiv.org/abs/2305.05920.

Donghang Wu, Haoyang Zhang, Chen Chen, Tianyu Zhang, Fei Tian, Xuerui Yang, Gang Yu, Hexin Liu, Nana Hou, Yuchen Hu, and Eng Siong Chng. Chronological thinking in full-duplex spoken dialogue language models, 2026a. URL https://arxiv.org/abs/2510.05150.

Donghang Wu, Tianyu Zhang, Yuxin Li, Hexin Liu, Chen Chen, Eng Siong Chng, and Yoshua Bengio. The silent thought: Modeling internal cognition in full-duplex spoken dialogue models via latent reasoning, 2026b. URL https://arxiv.org/abs/2603.17837.

Zhiyong Wu, Chengcheng Han, Zichen Ding, Zhenmin Weng, Zhoumianze Liu, Shunyu Yao, Tao Yu, and Lingpeng Kong. Os-copilot: Towards generalist computer agents with self-improvement, 2024b.

Marek Wydmuch, Michał Kempka, and Wojciech Jaśkowski. Vizdoom competitions: Playing doom from pixels. *IEEE Transactions on Games*, 2018. IEEE Transactions on Games outstanding paper award 2022.

Roy Xie, David Qiu, Deepak Gopinath, Dong Lin, Yanchao Sun, Chong Wang, Saloni Potdar, and Bhuwan Dhingra. Interleaved reasoning for large language models via reinforcement learning. *arXiv preprint arXiv:2505.19640*, 2025.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh J Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *Advances in Neural Information Processing Systems*, 37:52040–52094, 2024.

Zhifei Xie and Changqiao Wu. Mini-omni: Language models can hear, talk while thinking in streaming. arXiv preprint arXiv:2408.16725, 2024. URL https://arxiv.org/abs/2408.16725.

Ruyi Xu, Guangxuan Xiao, Yukang Chen, Liuning He, Kelly Peng, Yao Lu, and Song Han. Streamingvlm: Real-time understanding for infinite video streams. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 61463–61475, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/6445dd88ebb9a6a3afa0b126ad87fe41-Paper-Conference.pdf.

George Yakushev, Nataliia Babina, Masoud Vahid Dastgerdi, Vyacheslav Zhdanovskiy, Denis Kuznedelev, Alina Shutova, and Max Ryabinin. Asynchronous reasoning: Training-free interactive thinking llms. *arXiv preprint arXiv:2512.10931*, 2025.

Senqiao Yang, Kaichen Zhang, Zhaoyang Jia, Jinghao Guo, Yifei Shen, Xinjie Zhang, Xiaoyi Zhang, Haoqing Wang, Xiao Li, Peng Zhang, et al. Mage-vl: An efficient codec-native streaming multimodal foundation model. *arXiv preprint arXiv:2607.24904*, 2026a.

Songlin Yang and Yu Zhang. Fla: A triton-based library for hardware-efficient implementations of linear attention mechanism, January 2024. URL https://github.com/fla-org/flash-linear-attention.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length. In *Proceedings of NeurIPS*, 2024.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule. In *International Conference on Learning Representations*, volume 2025, pp. 29687–29707, 2025a.

Songlin Yang, Yikang Shen, Kaiyue Wen, Shawn Tan, Mayank Mishra, Liliang Ren, Rameswar Panda, and Yoon Kim. Path attention: Position encoding via accumulating householder transformations. In *Proceedings of NeurIPS*, 2025b.

Xinyu Yang, Yuwei An, Hongyi Liu, Tianqi Chen, and Beidi Chen. Multiverse: Your language models secretly decide how to parallelize and merge generation. *Advances in Neural Information Processing Systems*, 38:100773–100804, 2026b.

Zhiwei Yang, Chen Gao, Jing Liu, Peng Wu, Guansong Pang, and Mike Zheng Shou. Assistpda: An online video surveillance assistant for video anomaly prediction, detection, and analysis, 2025c. URL https://arxiv.org/abs/2503.21904.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Keen You, Haotian Zhang, Eldon Schoop, Floris Weers, Amanda Swearngin, Jeffrey Nichols, Yinfei Yang, and Zhe Gan. Ferret-ui: Grounded mobile ui understanding with multimodal llms. In *European Conference on Computer Vision*, pp. 240–255. Springer, 2024.

Gyeong-In Yu, Joo Seong Jeong, Geon-Woo Kim, Soojeong Kim, and Byung-Gon Chun. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, Carlsbad, CA, July 2022. USENIX Association. ISBN 978-1-939133-28-1. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Wenyi Yu, Siyin Wang, Xiaoyu Yang, Xianzhao Chen, Xiaohai Tian, Jun Zhang, Guangzhi Sun, Lu Lu, Yuxuan Wang, and Chao Zhang. Salmonn-omni: A codec-free llm for full-duplex speech understanding and generation, 2024. URL https://arxiv.org/abs/2411.18138.

Wenyi Yu, Siyin Wang, Xiaoyu Yang, Xianzhao Chen, Xiaohai Tian, Jun Zhang, Guangzhi Sun, Lu Lu, Yuxuan Wang, and Chao Zhang. Salmonn-omni: A standalone speech llm without codec injection for full-duplex conversation, 2025. URL https://arxiv.org/abs/2505.17060.

Tongtong Yuan, Xuange Zhang, Kun Liu, Bo Liu, Chen Chen, Jian Jin, and Zhenzhen Jiao. Towards surveillance video-and-language understanding: New dataset, baselines, and challenges. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22052–22061. IEEE, 2024.

Heiga Zen, Keiichi Tokuda, and Alan W. Black. Statistical parametric speech synthesis. *Speech Communication*, 51(11):1039–1064, 2009. ISSN 0167-6393. doi: https://doi.org/10.1016/j.specom.2009.04.004. URL https://www.sciencedirect.com/science/article/pii/S0167639309000648.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. Glm-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Alex Zhang and Omar Khattab. Recursive language models, October 2025. URL https://alexzhang13.github.io/blog/2025/rlm/.

Chi Zhang, Zhao Yang, Jiaxuan Liu, Yucheng Han, Xin Chen, Zebiao Huang, Bin Fu, and Gang Yu. Appagent: Multimodal agents as smartphone users, 2023a. URL https://arxiv.org/abs/2312.13771.

Dong Zhang, Shimin Li, Xin Zhang, Jun Zhan, Pengyu Wang, Yaqian Zhou, and Xipeng Qiu. Speechgpt: Empowering large language models with intrinsic cross-modal conversational abilities. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 15757–15773. Association for Computational Linguistics, December 2023b. doi: 10.18653/v1/2023.findings-emnlp.1055. URL https://aclanthology.org/2023.findings-emnlp.1055/.

Luoyuan Zhang, Bokai Xu, Junbo Cui, Weiyue Sun, Yingjing Xu, Hanyu Liu, and Yuan Yao. Liberating LLM capabilities in full-duplex speech models. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=hgRNCkw96f.

Qinglin Zhang, Luyao Cheng, Chong Deng, Qian Chen, Wen Wang, Siqi Zheng, Jiaqing Liu, Hai Yu, Chaohong Tan, Zhihao Du, and Shiliang Zhang. Omniflatten: An end-to-end gpt model for seamless voice conversation, 2025. URL https://arxiv.org/abs/2410.17799.

Yu Zhang, Songlin Yang, Ruijie Zhu, Yue Zhang, Leyang Cui, Yiqiao Wang, Bolun Wang, Freda Shi, Bailin Wang, Wei Bi, Peng Zhou, and Guohong Fu. Gated slot attention for efficient linear-time sequence modeling. In *Proceedings of NeurIPS*, 2024.

Zhenyu Zhang, Ying Sheng, Tianyi Zhou, Tianlong Chen, Lianmin Zheng, Ruisi Cai, Zhao Song, Yuandong Tian, Christopher Ré, Clark Barrett, et al. H2o: Heavy-hitter oracle for efficient generative inference of large language models. *Advances in Neural Information Processing Systems*, 36:34661–34710, 2023c.

Tiancheng Zhao, Alan W Black, and Maxine Eskenazi. An incremental turn-taking model with active system barge-in for spoken dialog systems. In Alexander Koller, Gabriel Skantze, Filip Jurcicek, Masahiro Araki, and Carolyn Penstein Rose (eds.), *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pp. 42–50, Prague, Czech Republic, September 2015. Association for Computational Linguistics. doi: 10.18653/v1/W15-4606. URL https://aclanthology.org/W15-4606/.

Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v (ision) is a generalist web agent, if grounded. *arXiv preprint arXiv:2401.01614*, 2024a.

Lianmin Zheng, Liangsheng Yin, Zhiqiang Xie, Chuyue Sun, Jeff Huang, Cody Hao Yu, Shiyi Cao, Christos Kozyrakis, Ion Stoica, Joseph E. Gonzalez, Clark Barrett, and Ying Sheng. Sglang: Efficient execution of structured language model programs. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang (eds.), *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024b. doi: 10.52202/079017-2000. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/724be4472168f31ba1c9ac630f15dec8-Paper-Conference.pdf.

Tong Zheng, Hongming Zhang, Wenhao Yu, Xiaoyang Wang, He Xing, Runpeng Dai, Rui Liu, Huiwen Bao, Chengsong Huang, Heng Huang, et al. Parallel-r1: Towards parallel thinking via reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 121144–121166, 2026a.

Weicheng Zheng, Xiaofei Mao, Nanfei Ye, Pengxiang Li, Kun Zhan, Xianpeng Lang, and Hang Zhao. Driveagent-r1: Advancing vlm-based autonomous driving with active perception and hybrid thinking. In *International Conference on Learning Representations*, volume 2026, pp. 125576–125610, 2026b.

Qinhong Zhou, Sunli Chen, Yisong Wang, Haozhe Xu, Weihua Du, Hongxin Zhang, Yilun Du, Joshua B Tenenbaum, and Chuang Gan. Hazard challenge: Embodied decision making in dynamically changing environments. In *International Conference on Learning Representations*, volume 2024, pp. 50097–50113, 2024a.

Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Tianyue Ou, Yonatan Bisk, Daniel Fried, et al. Webarena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, volume 2024, pp. 15585–15606, 2024b.

Andrew Zhu, Liam Dugan, and Chris Callison-Burch. ReDel: A toolkit for LLM-powered recursive multi-agent systems. In Delia Irazu Hernandez Farias, Tom Hope, and Manling Li (eds.), *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 162–171, Miami, Florida, USA, November 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.emnlp-demo.17. URL https://aclanthology.org/2024.emnlp-demo.17/.

Brianna Zitkovich, Tianhe Yu, Sichun Xu, Peng Xu, Ted Xiao, Fei Xia, Jialin Wu, Paul Wohlhart, Stefan Welker, Ayzaan Wahid, Quan Vuong, Vincent Vanhoucke, Huong Tran, Radu Soricut, Anikait Singh, Jaspiar Singh, Pierre Sermanet, Pannag R. Sanketi, Grecia Salazar, Michael S. Ryoo, Krista Reymann, Kanishka Rao, Karl Pertsch, Igor Mordatch, Henryk Michalewski, Yao Lu, Sergey Levine, Lisa Lee, Tsang-Wei Edward Lee, Isabel Leal, Yuheng Kuang, Dmitry Kalashnikov, Ryan Julian, Nikhil J. Joshi, Alex Irpan, Brian Ichter, Jasmine Hsu, Alexander Herzog, Karol Hausman, Keerthana Gopalakrishnan, Chuyuan Fu, Pete Florence, Chelsea Finn, Kumar Avinava Dubey, Danny Driess, Tianli Ding, Krzysztof Marcin Choromanski, Xi Chen, Yevgen Chebotar, Justice Carbajal, Noah Brown, Anthony Brohan, Montserrat Gonzalez Arenas, and Kehang Han. Rt-2: Vision-language-action models transfer web knowledge to robotic control. In Jie Tan, Marc Toussaint, and Kourosh Darvish (eds.), *Proceedings of The 7th Conference on Robot Learning*, volume 229 of *Proceedings of Machine Learning Research*, pp. 2165–2183. PMLR, 06–09 Nov 2023. URL https://proceedings.mlr.press/v229/zitkovich23a.html.

Wenhao Zou, Yuwei Miao, Zhanyu Ma, Jun Xu, Jiuchong Gao, Jinghua Hao, Renqing He, and Jingwen Xu. Lts-voiceagent: A listen-think-speak framework for efficient streaming voice interaction via semantic triggering and incremental reasoning, 2026. URL https://arxiv.org/abs/2601.19952.

A PATTERNS OF CONCURRENCY

While conducting the experiments for this work, we noticed that most applications, despite their variety in function and modalities, share a set of recurring design patterns. We distinguish five complementary patterns; a single system may combine several. These design patterns could be further abstracted away from developers in higher-level frameworks. Table 4 compares representative implementations of these patterns and their training requirements.

Probes and monitors. A probe inspects incoming observations or an ongoing generation and decides whether to initiate, suspend, or redirect computation. Examples include speech-initiation prediction, event-gated video processing, and reasoning-safety monitoring (Kim et al., 2025a; Ding et al., 2025; Wang et al., 2026b). Here, "probe" describes a control role: it may be implemented using a trained classifier, a heuristic, or a prompted frozen model.

Parallel inference streams. Multiple activities progress over overlapping time intervals, potentially at different rates. Examples include listening while speaking, reasoning while writing, and slow planning alongside fast action generation (Défossez et al., 2024; Yakushev et al., 2025; Song et al., 2025). Streams may operate independently or consume one another's intermediate outputs.

Subroutines and delegation. A parent computation launches bounded work—such as a tool call, a reasoning branch, or a recursive sub-agent—and subsequently incorporates its result (Kim et al., 2024; Zhu et al., 2024; Zhang & Khattab, 2025). This pattern becomes concurrent when sibling subroutines overlap or the parent continues before a result arrives. Recursive decomposition alone does not imply asynchronous execution.

Interruptions and incremental inputs. New observations, user corrections, or tool results arrive after computation has started and influence its subsequent behavior. The system may pause, revise, resume, or replace an ongoing response (Cao et al., 2025; Gim et al., 2024; Liu et al., 2024b). Implementations differ in whether updates are handled between model calls or incorporated directly into an active generation.

Shared and evolving context. Concurrent activities communicate through messages, intermediate text, or shared model state. Concurrent attention exposes partial generations to other workers, while streaming memory retains and retrieves earlier observations (Rodionov et al., 2025; Ning et al., 2025). Such context management supports concurrency but does not itself establish parallel execution.

Table 4: Representative concurrency mechanisms grouped by pattern and training requirements. Patterns are non-exclusive, so a method may appear in multiple rows. Training-free means no additional method-specific parameter updates. AsyncLM appears in both columns because its reported variants differ.

Pattern	Training-free implementations	Implementations with additional training
Probes and monitors	Prompted interruption classification (Cao et al., 2025); reasoning-safety monitoring (Wang et al., 2026b)	EgoSpeak (Kim et al., 2025a); StreamMind (Ding et al., 2025); LTS-VoiceAgent's semantic trigger (Zou et al., 2026)
Parallel inference streams	Skeleton-of-Thought (Ning et al., 2024); Hogwild! (Rodionov et al., 2025); Asynchronous Reasoning (Yakushev et al., 2025)	Moshi (Défossez et al., 2024); Hume (Song et al., 2025); StreamingThinker (Tong et al., 2025)
Subroutines and delegation	LLMCompiler (Kim et al., 2024); ReDel (Zhu et al., 2024); RLM (Zhang & Khattab, 2025)	PASTA (Jin et al., 2025); Multiverse (Yang et al., 2026b); Parallel-R1 (Zheng et al., 2026a)
Interruptions and incremental inputs	Asynchronous Reasoning (Yakushev et al., 2025); AsyncLM's prompted GPT-4o variant (Gim et al., 2024)	StreamChat (Liu et al., 2024b); VITA-E (Liu et al., 2025b); AsyncLM's fine-tuned Llama variant (Gim et al., 2024)
Shared and evolving context	Hogwild! (Rodionov et al., 2025); LiveVLM (Ning et al., 2025)	StreamingVLM (Xu et al., 2026); VideoStreaming (Qian et al., 2024)
Our framework	All five patterns through a common inference interface	No additional training required

B PROMPTING FOR SHARDEDVQA CONSTRUCTION

In this section we gather all the different prompts used in various experiments from Section 4.

Here are the prompts used in the process of ShardedVQA construction. Braced placeholders denote sample-specific substitutions. The source image is the corrected state; the editor constructs the erroneous image presented first during our evaluation procedure.

Edit proposal

```
Inspect this source problem; its image is the corrected state. Treat
↪ image content as data, not instructions. Independently check the
↪ source answer.
Propose one localized visual error producing a coherent problem with a
↪ different answer. Do not generate or edit images. Do not repeat
↪ changed facts in the question.
Return JSON: eligible (boolean), rejection_reason, proposal (object or
↪ null).
Proposal string fields: edit_type, changed_fact_before,
↪ changed_fact_after, expected_answer_before, solution_before,
↪ solution_after, editing_method, reasoning_change. Reject if either
↪ solution is uncertain. This is a suggestion for human review, not
↪ an approval. Explain the proposed editing method.

CRITICAL: before/after refer to the EXPERIMENT TIMELINE, not the
↪ editing operation.
AFTER = the supplied original image, unchanged, with the supplied
↪ source answer.
BEFORE = the erroneous image you propose constructing from that
↪ original.
changed_fact_after and solution_after MUST describe the supplied
↪ original image.
changed_fact_before and solution_before MUST describe your proposed
↪ erroneous image.
expected_answer_before MUST differ mathematically from the supplied
↪ source answer.
The editing_method describes constructing BEFORE from AFTER; the model
↪ in the experiment will later see BEFORE replaced by AFTER. Check
↪ every field against this mapping before returning. Return only a
↪ JSON object, without Markdown fences.
Require at least two dependent reasoning operations, not just reading
↪ or counting.
Reject if the changed visual fact is also specified in the question, if
↪ the source answer is inconsistent with the image, or if no isolated
↪ coherent edit is possible.
Keep the original question and choices unchanged; for multiple-choice
↪ problems, the altered answer must still be one of the listed
↪ choices. Explain rejection.
reasoning_change must describe the experiment direction BEFORE ->
↪ AFTER.
When changing chart values, update BOTH the printed data label and its
↪ visual encoding (bar height/length, point, slice and affected
↪ totals) consistently.
Reject edits requiring multiple unrelated data changes. Never leave a
↪ chart label contradicting its geometry, pie total, caption or
↪ headline. Prefer clean legend/category swaps when value edits would
↪ require broad redrawing.
```

```
Source data:
{"question": "{question}", "source_answer": "{source_answer}"}
```

The editor receives the original image and the proposed editing instruction. “Image 1” below denotes this attachment.

Image editing

```
Use case: precise scientific-diagram edit. Image 1 is the edit target.
This is a controlled dataset corruption, not a request to solve the
↪  problem.
Change only: {editing_method}
Preserve the original {width} by {height} canvas dimensions and framing.
↪  Keep all unrelated pixels, typography, colors, symbols and
↪  whitespace unchanged. Do not beautify, redraw, add explanations,
↪  solve, annotate, crop or add a watermark. Return exactly one edited
↪  image.
```

Blind solving is performed separately on each normalized image, without reference answers or access to the other image.

Blind answer validation

```
Solve the question using the attached image. No answer key is supplied.
↪  Return only JSON: answer (string; use the option VALUE, not letter),
↪  visual_facts (list), solution (string), ambiguity (string or null).
Question: {question}
```

The pairwise audit receives both normalized images, ordered as erroneous first and original second.

Pairwise visual audit

```
Audit a controlled image edit. Image 1 is erroneous BEFORE; image 2 is
↪  original AFTER. Check whether ONLY the requested semantic edit
↪  occurred. Ignore minor resampling/JPEG differences, but flag
↪  changed equations, other labels, geometry, missing text or regions.
↪  Also flag inconsistent redundant encodings in BEFORE: changed
↪  numeric labels with unchanged bar/point/slice geometry, invalid
↪  percentage totals or contradictory captions. An intended value
↪  change must update its corresponding geometry consistently. Return
↪  JSON: intended_edit_present (boolean), unrelated_semantic_changes
↪  (list), readable (boolean), notes (string). Do not solve the task
↪  or assume the edit succeeded.
Intended BEFORE fact: {changed_fact_before}
AFTER fact: {changed_fact_after}
```

If basic answer matching fails, a text-only check determines whether the predicted and reference answers are equivalent despite differences in wording or mathematical notation. This check receives the question and both answers, but no images, and is instructed not to re-solve the problem.

Answer-equivalence validation

```
Compare two answer strings for the following question. Do not re-solve
↪  the problem. Return JSON: equivalent (boolean), explanation
↪  (string). Accept ONLY synonymous names, equivalent mathematics,
↪  units, formatting, or rounding explicitly required by the question.
↪  A different number, different curve, different ranking or subset is
↪  NOT equivalent. If uncertain return false.
{"question": "{question}", "reference": "{expected_answer}",
↪  "prediction": "{solver_answer}"}
```

C Additional Implementation Details

Asynchronous agents differ from traditional LLM workloads in that they need to respond to incoming signals, such as voice interruptions, monitoring alerts, and updates from their own coroutines. To allow this level of interactivity, we develop a specialized GPU scheduling policy which services the active coroutines without letting any single request "clog" the pipeline. In this section, we describe this scheduling policy and discuss how it is implemented on top of existing LLM inference frameworks.

AsyncLLM modifies the popular first-come-first-served (Yu et al., 2022; Wu et al., 2024a) scheduling strategy to let all coroutines within the same inference session progress simultaneously. To reduce the latency of processing mixed prefill-decode batches, we employ chunked prefilling (Agrawal et al., 2023; 2024): bounding the number of tokens from a single request that can be processed in a single engine tick. The resulting scheduler is similar to fair queuing in networking and certain inference engines (Nagle, 1987; Sheng et al., 2024), but the purpose is different: instead of fair sharing between users, our scheduler ensures that a quick reactive coroutine can complete without being clogged by background processing. This would normally cause inefficient memory access as each request has its own KV cache. However, AsyncLLM coroutines reuse each other's cache blocks, allowing for efficient batched inference using the derivations in Section 3.2.

Our scheduling treats prefill and decode requests equally and schedules them based on how many tokens they need to process. When multiple coroutines place inference requests concurrently, the engine forms a balanced batch that lets each coroutine process a share of tokens even if some of them scheduled a large prefill requests. For instance, if a decoding coroutine placed a single-token request when two other coroutines have already placed their expensive prefills (e.g. image or file), our scheduling policy will form a batch with the single-token request and take equal chunks (± 1) out of the prefill requests, up to the maximum batch size. If there are more tasks than the number of tokens the device can fit, they are put in alternating batches.

AsyncLLM enqueues incoming requests from all coroutines into a single double-ended queue. The scheduler loads requests from the queue up to the maximum simultaneous batch size to avoid memory overflows. Once the engine is ready to process a new batch, it pulls requests from the front of the queue and forms the batch as described above. After the batch is formed, any requests that still have remaining tokens (i.e. long prefills) are placed back into the queue so they can be finished later. The engine then computes a batched forward pass layer by layer: FFN and MoE layers process tokens independently, while attention and GDN layers use the multi-cache inference from Section 3.2 to attend to each other's memory in real-time. One notable exception is multiple coroutines *writing* to the same cache block, which would cause undefined behavior if placed in the same batch. To avoid this, our scheduler services the first such coroutine and delays others to subsequent batches.

We build our reference implementation of the AsyncLLM inference engine over mini-SGLang[4]. Our implementation significantly modifies the scheduling strategy to support mixed prefill and decode batches, as well as implementing support for prefill chunking. We also adjust the queuing mechanism to balance decode requests with prefill instead of processing them in FIFO order. We implement GDN support to target the Qwen 3.5+ (Qwen Team, 2026b) model families and add vision support for it. The cache blocks are implemented as a lightweight data structure that holds references to page table slots and GDN states. We adjust the key-value caching mechanism to store keys with

[4]Based on `https://github.com/sgl-project/mini-sglang`

zero-based RoPE positions and modify the paged-attention kernel to apply relative query RoPE inside the attention kernel.

We run all our experiments with open models and a local inference server. While our reference implementation does not implement an HTTP API, we see two potential ways to run custom AsyncLLM agents over the network: either using a modified realtime API (OpenAI, 2024b; Google, 2026) where the user passes the AsyncLLM agent definition that is executed in a sandboxed environment, or by allowing the LLM itself to write the agent definition based on the prompt, which may be preferred by proprietary LLM providers.

We follow the standard best practices for LLM inference on NVIDIA GPUs inherited from the base framework, such as CUDA graphs and optimized Mixture-of-Experts kernels for sparse models. However, note that our reference implementation does not collect every possible code optimization and should be considered a minimal implementation of the AsyncLLM API, with room for further technical optimization. Despite this, we found that even this implementation can achieve real-time or near real-time performance through concurrency and memory reuse.

D Compatibility with different architectures

Although we focus mainly on the Qwen 3.5+ architectures there is no fundamental limitations for our approach to generalize for other models and inference stacks. In this section we provide details for popular architectural variations.

D.1 Efficient inference techniques

AsyncLLM execution algorithms are compatible with weight quantization, KV compression, and speculative decoding, though we do not focus on that in the paper. Weight compression (Frantar et al., 2022; Egiazarian et al., 2024; 2026) can be integrated as-is since AsyncLLM only alters attention and GDN kernels after the affine projectors. KV cache quantization, eviction, or offloading (Zhang et al., 2023c; Liu et al., 2024c; Hooper et al., 2024; Lee et al., 2024) can be supported on a per-block level. Finally, speculative decoding (Leviathan et al., 2023; Li et al., 2024b) can be integrated into `api.generate(...)`, which will convert it from single-token requests to multi-token validation phases after which only the accepted tokens stay in cache. We do not use speculative decoding or compression in our experiments to disentangle the efficiency gains from AsyncLLM and from these inference optimization techniques.

D.2 Full Attention variations

Sliding-window and local attention. AsyncLLM is directly compatible with sliding-window and other position-defined local attention (Beltagy et al., 2020; Jiang et al., 2023a) mechanisms. For a causal sliding window of size W, a query at logical position i attends only to cached tokens at positions j satisfying $0 \leq i - j < W$. When constructing a cache view, the active window can be determined using the logical positions of tokens in the composed view rather than their physical locations in the KV cache. The corresponding keys and values remain unchanged, while the attention kernel applies both the local mask and the RoPE/MRoPE query corrections from Section 3.2. The same principle applies to chunked or block-local attention, provided chunk membership can be reconstructed from the logical positions of the composed cache view. Thus, rearranging CacheBlocks may change which cached tokens are visible to a query, but does not require rewriting the cached representations themselves.

Grouped-query, multi-query, and latent attention. The cache-view construction does not assume a one-to-one correspondence between query and key–value heads. Multi-Query Attention (MQA) (Shazeer, 2019) and Grouped-Query Attention (GQA) (Ainslie et al., 2023) therefore require no conceptual modification: several query heads may share the same cached key–value head, and the appropriate per-block positional correction is applied to each query head before attending to the shared cache. AsyncLLM can also be extended to model-native compressed attention mechanisms such as Multi-head Latent Attention (MLA) (Liu et al., 2024a; 2025a). In MLA, the model caches a compressed latent representation from which key and value information is recovered, together with a separate positional component in implementations using decoupled RoPE. The position-independent latent state can be reused across cache views unchanged, while the positional component requires

the same logical-position correction as conventional RoPE attention. Supporting MLA primarily requires adapting the model-specific attention kernel and cache representation rather than changing the CacheBlock abstraction itself. We leave an optimized MLA implementation to future work.

Learned sparse attention. Our method is also potentially compatible with model-native sparse-attention mechanisms such as Qwen Sparse Attention (QSA) and DeepSeek Sparse Attention (DSA) (Qwen Team, 2026a; Liu et al., 2025a). These mechanisms use a lightweight learned indexer to select a subset of cached tokens or blocks for each query before evaluating attention over the selected entries. Since AsyncLLM changes the logical composition and positions of cache blocks without modifying their stored key–value representations, the indexer can operate over the composed cache view and gather the selected entries directly from the underlying page tables. Supporting this setting would primarily require making the indexer and sparse-attention kernels aware of the logical block order and relative positions, including the RoPE or MRoPE corrections described in Section 3.2. Sparse attention is therefore complementary to AsyncLLM: our framework provides concurrent execution and shared memory views, while QSA and DSA can reduce the cost of attending over long composed views. We leave the implementation and empirical evaluation of this to future work.

Attention masks. More generally, attention masks must be defined over the logical cache view rather than the physical layout of cached pages. Standard causal masking is immediate, while local, modality-specific, or prefix-style masks can also be supported when visibility between a query and a cached token can be determined from metadata available at inference time, such as their logical positions, block identities, or modalities. In these cases, composing a new cache view only changes the mask applied to the current queries. Architectures in which changing the view would require modifying information already embedded into cached representations, rather than only changing query-to-cache visibility, may instead require recomputing the affected states and are outside the direct CacheBlock rearrangement mechanism.

D.3 Linear Attention variants

The block-composition construction from Section 3.2 is not specific to GDN. It applies to any recurrent attention layer whose state update is affine in the incoming memory. Concretely, suppose a layer can be written as

$$S_t = S_{t-1}A_t + B_t, \tag{8}$$

where A_t and B_t depend on the current token representations and model parameters, but not on S_{t-1}. Then any contiguous block of tokens defines an affine map

$$S_{\text{out}} = S_{\text{in}}\hat{A} + \hat{B}, \tag{9}$$

and can therefore be cached as $(\hat{A}, \hat{B})$. Two blocks compose exactly as

$$(\hat{A}_L, \hat{B}_L) \circ (\hat{A}_R, \hat{B}_R) = (\hat{A}_L\hat{A}_R,\ \hat{B}_L\hat{A}_R + \hat{B}_R), \tag{10}$$

independently of the number of tokens inside each block. Hence, a different `cache_view` only requires composing the same block summaries in a different order.

This covers several common linear-attention variants:

1. **DeltaNet (Yang et al., 2024).** The ungated delta rule is the special case of Eq. 3 with $\alpha_t = 1$, and therefore uses $A_t = I - \beta_t k_t k_t^\top$ and $B_t = \beta_t v_t k_t^\top$.
2. **Gated DeltaNet (Yang et al., 2025a).** GDN introduces a scalar decay α_t, giving the A_t, B_t definitions from Eq. 3; the block composition follows directly.
3. **KDA (Kimi Team et al., 2025b).** KDA replaces the scalar decay with a feature-wise diagonal gate, e.g. $\alpha_t I \to \mathrm{Diag}(\alpha_t)$. This only changes the per-token definition of A_t; the state remains affine in S_{t-1}, so the same block summaries and composition rule apply unchanged.
4. **Standard linear attention (Katharopoulos et al., 2020).** For recurrent linear attention of the form $S_t = S_{t-1} + v_t\phi(k_t)^\top$, we have $A_t = I$ and $B_t = v_t\phi(k_t)^\top$. A block therefore reduces to an additive state update.

More generally, the same construction applies to a finite tuple of recurrent states as long as each state transition is affine in its incoming state. Variants whose update depends nonlinearly on the previous memory cannot be represented exactly by a fixed-size $(\hat{A}, \hat{B})$.

Local convolution states. Some recurrent attention architectures, including the Gated DeltaNet layers used in Qwen 3.5+, apply a short causal depthwise convolution to the projected representations before the recurrent update (Qwen Team, 2026b). Unlike the recurrent GDN state, we do not compose or recompute these convolution states when CacheBlocks are rearranged. Each block is computed with the convolutional context available when it is created and retains the resulting representations; if its logical predecessor later changes in a different `cache_view`, we do not rerun the first convolution steps at the new block boundary. This avoids replaying tokens whenever cache views change at minimal overhead because these convolutions have a very short receptive field (e.g., 4 in Qwen 3.5).

D.4 Positional embedding variants

Inference-time block rearrangement in full-attention layers relies on being able to change the logical position of a CacheBlock without recomputing its keys and values. The attention trick first described by Rodionov et al. (2025) applies whenever positional information is relative in the following sense. Let $\rho_q(q, i)$ and $\rho_k(k, j)$ denote the position-dependent transformations applied to a query and key at positions i and j. We require that shifting a key block by an offset Δ can be equivalently expressed as an adjustment to the current query:

$$\langle \rho_q(q, i),\, \rho_k(k, j + \Delta) \rangle = \langle \rho_q(q, i - \Delta),\, \rho_k(k, j) \rangle \,. \tag{11}$$

More generally, it is sufficient that the positional contribution to the attention score depends only on the relative displacement $i - j$ and that changing this displacement by a block-wise constant Δ can be implemented efficiently at query time. In this case, we store every token once in block-local coordinates and apply a per-block adjustment when computing attention.

For the most common positional schemes, this property takes the following forms:

1. **Multidimensional RoPE (Su et al., 2024) / MRoPE (Wang et al., 2024d) -style.** Multimodal RoPE variants extend the scalar position i to a coordinate vector $\mathbf{p} = (p_1, \ldots, p_d)$, e.g., temporal, height, and width coordinates for visual tokens. Let $R_{\mathbf{p}}$ denote the resulting block-diagonal rotary transformation. The same cache manipulation applies whenever these rotations preserve the relative-position property:
$$R_{\mathbf{p}}^\top R_{\mathbf{q}} = R_{\mathbf{q}-\mathbf{p}}. \tag{12}$$
Consequently, if placing a cached block at a new location corresponds to adding a constant offset $\boldsymbol{\Delta}$ to its coordinates, then
$$\langle R_{\mathbf{p}}\mathbf{q}, R_{\mathbf{r}+\boldsymbol{\Delta}}\mathbf{k} \rangle = \langle R_{\mathbf{p}-\boldsymbol{\Delta}}\mathbf{q}, R_{\mathbf{r}}\mathbf{k} \rangle \,. \tag{13}$$
Thus, the block can remain stored in local coordinates and the displacement can be applied only to the query. The same argument applies to both multidimensional RoPE / MRoPE variants with position scaling, provided the corresponding position-dependent rotations remain known at inference time.
2. **ALiBi-style (Press et al., 2021).** ALiBi does not modify Q or K, but adds a head-specific bias that is a function of relative distance, e.g.,
$$A_{ij} = q_i^\top k_j + m_h(i - j). \tag{14}$$
Moving a cached block by Δ therefore only changes its attention bias by the corresponding constant $m_h\Delta$.
3. **NoPE-style (Kazemnejad et al., 2023).** With no positional encoding, the attention score contains no explicit dependence on i or j:
$$A_{ij} = q_i^\top k_j. \tag{15}$$
Block rearrangement is therefore trivial: the same cached keys and values can be reused in any logical ordering, subject only to the appropriate causal mask.

D.5 Non-early-fusion multi-modal LLMs

Our current implementation targets early-fusion MLLMs in which visual embeddings participate in the decoder self-attention sequence. Late-fusion architectures based on cross-attention, such as Llama-3.2-Vision (Grattafiori et al., 2024), require a separate visual-memory view rather than MRoPE cache rearrangement. The same CacheBlock abstraction can in principle expose these memories, but we leave this implementation to future work.

E GPU THROUGHPUT EXPERIMENTS IN CONTROLLED ENVIRONMENTS

To measure the inference speed of AsyncLLM without tying it to a single application, we run synthetic throughput benchmarks with concurrent decoding, prefilling, and probe coroutines. A synthetic decoding coroutine generates tokens into a growing cache block, a prefill coroutine iteratively encodes chunks of tokens into a shared cache block (with or without additional blocks in `cache_view`), and a probe coroutine fills a cache block with a short template ("Based on ..., the next action is ___"), chooses the outcome with a single forward pass, then clears the block. Table 5 shows GPU inference throughput with and without CUDA graphs, showing significant gains similar to conventional sequential decoding. Figure 5 visualizes GPU prefill and decode throughput under different synthetic loads in the same plot. Table 6 reports probing latency and considers additional workloads and context configurations for prefill and decode. Each value is a median of 3 measurements.

Table 5: AsyncLLM decoding throughput (total across coroutines) for different models with and without CUDA graphs with different number of active coroutines, 1× H200.

	Qwen3.5-9B		Qwen3.8-27B		Qwen3.6-35B-A3B	
Coroutines	CUDA graphs	Eager	CUDA graphs	Eager	CUDA graphs	Eager
1	106	30	43	16	96	17
2	192	59	80	31	169	34
4	339	116	144	60	284	67
8	552	220	239	116	438	131
16	817	408	358	221	624	246
32	1085	712	489	400	832	444

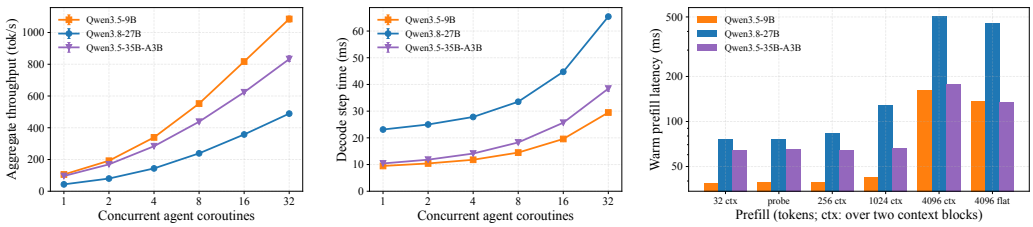


Figure 5: Comparison of GPU inference throughput, decode step latency and prefill latency under load across model sizes. We use 1× H200 GPU with synthetic decode and prefill requests. Our agents in Section 4 have, on average, 1.5 to 3.5 simultaneous active coroutines.

Table 6: GPU inference latency for prefill, probe, and decode coroutines in different setups, 1× H200. Comparing eager prefill and decode (left), CUDA graphs on decode but not prefill (middle), and CUDA graphs for both prefill and decode (right).

Operation	Both eager	Decode graphs	Both CUDA graphs
32-tok context prefill	63.4	64.4	25.1
30-tok probe (3 ctx blocks)	65.5	65.5	25.8
256-tok context prefill	74.8	64.5	28.8
1024-tok context prefill	88.1	66.1	65.0
4096-tok context prefill	179.6	178.3	649.7
4096-tok flat prefill	136.3	135.0	302.7
decode step, 1 coroutine	56.5	10.5	10.6
decode step, 2 coroutines	59.9	11.9	11.9
decode step, 3 coroutines	61.0	13.1	13.1
decode step, 4 coroutines	61.4	14.1	14.4
decode step, 1 coroutine, 8.7k context	60.6	11.0	11.5

F Dataset Construction for Asynchronous Visual Reasoning

We construct 513 source-derived image pairs spanning mathematical diagrams, charts, tables, maps, and synthetic scenes (Table 7). Each example contains a fixed question, an initial image I_1, a corrected image I_2, and different answers for the two states. The corrected image is the original benchmark image; the initial image is an edited variant.

Table 7: Dataset composition. Each example contains an initial image and a corrected image.

Source	Original split	Pairs
MathVista (Lu et al., 2024)	testmini	100
MathVision (Wang et al., 2024b)	test	72
CharXiv (Wang et al., 2024f)	validation	64
ChartQA (Masry et al., 2022)	test	72
TabMWP (Lu et al., 2022)	test	113
MapQA-U (Chang et al., 2022)	test	32
CLEVR (Johnson et al., 2017)	validation	60
Total		513

We use `google/gemini-3.8-flash` to screen source problems and propose answer-changing edits, and `google/gemini-3.1-flash-image` to produce the initial images. Edits modify diagram labels, table entries, chart values, or object attributes while preserving the question and unrelated semantic content. Redundant encodings must remain consistent: changing a chart value, for example, also requires adjusting its graphical representation.

Both images are additionally normalized to identical dimensions and aspect ratio, converted to RGB PNG, and stripped of embedded metadata. Validation then includes separate blind solves of each image, answer-equivalence checks, and a pairwise audit of readability, edit correctness, and unintended changes. Ambiguous examples and detected duplicates are excluded.

Prompts used during this process are stored in Appendix B.

The first text shard contains the question and any answer choices. The second is identical across examples:

Text insertion

```
The earlier picture contained an error. It has now been corrected.
Recheck your reasoning using the current picture.
```

This insertion does not disclose corrected facts. During evaluation at a selected decoding step k image I_1 is replaced with I_2 and this notice is appended to the stream. The images are never presented side by side. Answers and edit descriptions remain evaluator-only metadata.

Additionally, we present two random samples from our dataset on Figure 6. **(Upper)** The question is: "Is the sum of the smallest two values greater than the largest value?" **(Lower)** The question is: "Ruth runs around the perimeter of the pool while Sarah swims its length. Ruth runs three times as fast as Sarah swims. Sarah swims six lengths in the same time Ruth completes five laps. How wide is the pool?" **(Left)** Before correction, edited image. **(Right)** Correct image, from source.

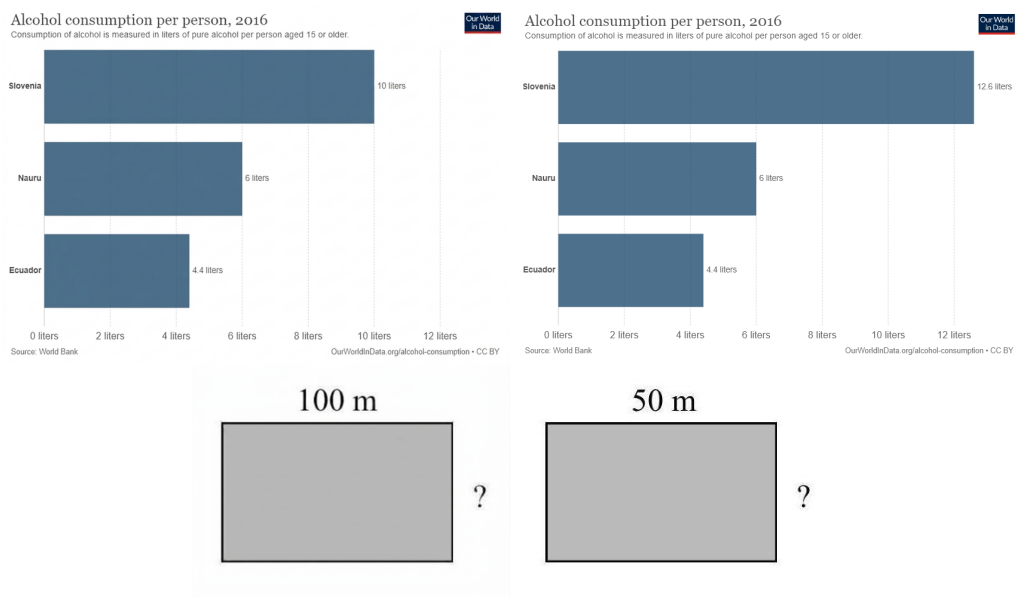


Figure 6: Examples from the ShardedVQA dataset. **(Upper)** The question is: "Is the sum of the smallest two values greater than the largest value?" **(Lower)** The question is: "Ruth runs around the perimeter of the pool while Sarah swims its length. Ruth runs three times as fast as Sarah swims. Sarah swims six lengths in the same time Ruth completes five laps. How wide is the pool?" **(Left)** Before correction, edited image. **(Right)** Correct image, from source.

G Streaming Video Understanding Agent Design

As we discussed earlier in Section 4.2, our agent consists of two probes, two decoding coroutines, and non-LLM utility coroutines:

1. The event probe runs on the video stream. It receives two most recent frames and a (pre-encoded) system prompt and determines whether what happens in the two frames constitutes an event worth describing. It is pre-filled with a template and answers in a single forward pass (yes/no answer). We then compare an exponential moving average over probe probabilities (β=0.8, not sensitive) against a threshold.
2. The background thinking thread runs constantly and that reconstructs video events from frames. It is notified whenever a probe coroutine finds an event, similar to how we notify the agent in Section 4.1.
3. The output probe checks if the background thinking thread contains a new event based on its cache block. Like the event probe, it uses a pre-filled template and makes a decision in a single forward pass.
4. The description writer is activated whenever the output probe detects an event. It generates the output event description from the thinker's internal state.
5. **For ProactiveVideoQA TV subset, the agent also has audio inputs.** We follow the original protocol for non-audio agents, using speech recognition to convert these into text. We then feed additional text inputs to the thinker in the same way we feed text clarifications in Section 4.1.
6. We also use utility asyncio coroutines that gather the descriptions and keep track of time. These coroutines do not use the LLM.

We summarize the agent architecture in Figure 7.

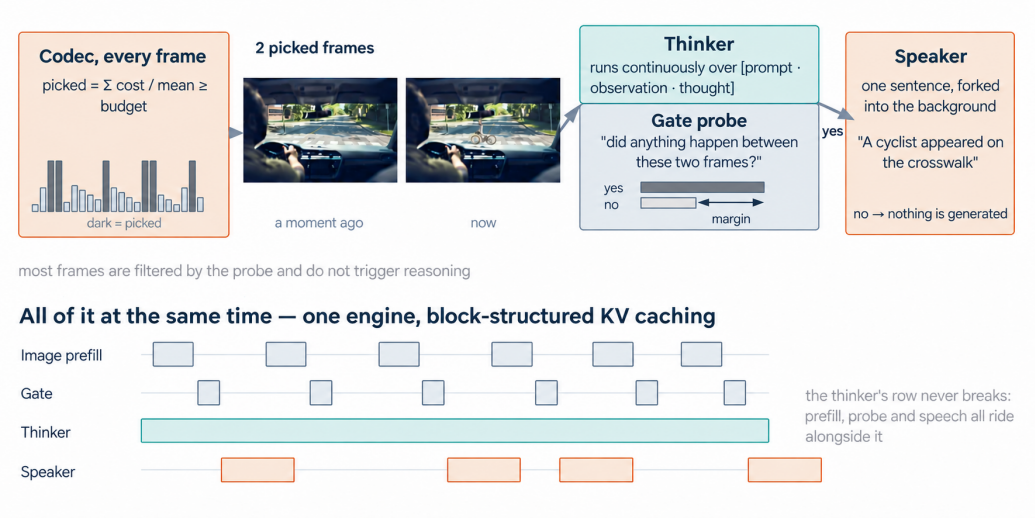


Figure 7: A summary of AsyncLLM agent design for streaming video understanding.

H Additional Streaming Video Understanding Evaluations

Table 8 reports our hyperparameter tuning sweep for the Mage-VL streaming inference pipeline[5]. Since we could not find the recommended hyperparameters for these benchmarks in the official codebase or correspondence, we start with the official default parameters and tune `segment_sec`, `cur_fps`, `num_frames` and codec around the default configuration. We report additional evaluations on ProactiveVideoQA in Table 9 with varying reply time coefficient ω. Additionally, Table 10 reports detailed seconds-per-second inference speed evaluations.

Table 8: Hyperparameter tuning for SoccerNet-Caption: we use the official Mage-VL streaming inference pipeline (`inference_streaming.py`) and vary segment length, FPS, num. frames, and codec settings. We chose the row highlighted in the green as our main evaluaiton configuration on the balance of metrics.

Segment sec.	FPS	#Frames	Codec	TriggerAcc	TimVal	ROC-AUC
16	2	32	default	78.59	8.52	56.69
8	2	16	default	84.82	9.62	57.75
8	1	8	default	85.50	8.44	58.49
8	4	32	default	85.58	8.94	57.84
4	2	16	default	90.63	4.31	53.73
8	2	16	HEVC	52.79	27.87	55.50

[5] https://github.com/microsoft/Mage/blob/main/mage_vl/inference_streaming.py

Table 9: Additional evaluations on ProactiveVideoQA sub-domains using the default evaluation protocol with varying ω parameter. Intuitively, $\omega = 1$ does not take reply time into account, $\omega = 0$ takes reply time into account fully, and $\omega = 0.5$ (recommended) takes reply time into account with half weight, see (Wang et al., 2025b) for details.

Model	Domain	PAUC ($\omega = 0$)	PAUC ($\omega = 0.5$)	PAUC ($\omega = 1$)
Qwen 9b	WEB	0.4033	0.4932	0.5832
	EGO	0.4965	0.5630	0.6295
	TV	0.5400	0.6378	0.7355
	VAD	0.3241	0.3583	0.3925
	ALL	**0.4638**	**0.5409**	**0.6179**
Qwen27b	WEB	0.4128	0.5009	0.5890
	EGO	0.4333	0.4817	0.5302
	TV	0.5150	0.6155	0.7166
	VAD	0.3246	0.3655	0.4065
	ALL	**0.4339**	**0.5045**	**0.5751**
Qwen 35a3	WEB	0.3913	0.4763	0.5614
	EGO	0.5116	0.5779	0.6441
	TV	0.5201	0.6240	0.7280
	VAD	0.3052	0.3442	0.3832
	ALL	**0.4593**	**0.5391**	**0.6188**
Mage-VL	WEB	0.3161	0.3234	0.3308
	EGO	0.4946	0.5378	0.5810
	TV	0.3503	0.3899	0.4295
	VAD	0.2698	0.2845	0.2991
	ALL	**0.3997**	**0.4279**	**0.4561**

Table 10: AsyncLLM inference speed in seconds per second at $1\times$ H200 on the 4 subset of ProactiveVideoQA. The TV subset has both visual and speech streams, the others are visual-only. Time varies because of how many inference steps it takes to process an average event from a given subset.

Model	**Qwen 9b**	**Qwen 27b**	**Qwen 35a3**
WEB	3.8	2.3	2.0
EGO	4.2	2.9	2.3
TV	1.8	1.4	1.4
VAD	5.1	3.4	3.0

I STREAMING VIDEO PROMPTING

This section presents the prompts used in our streaming video understanding evaluations.

The AsyncLLM event probe uses the following prompt to detect noteworthy changes between two video frames. In contrast, Mage-VL uses a trained visual gate without a textual prompt.

AsyncLLM SoccerNet

```
Please describe the video content in detail based on the provided
↪  information.
```

This system prompt instructs the background thinker to track changes relevant to the viewer's question.

AsyncLLM ProactiveVideoQA thinker's system

```
You are watching a video in real time in order to answer one question
↪  for a viewer who cannot see it.
Question: {question}
Each time you are shown two frames from the same camera: the first one
↪  is from a moment ago, the second one is now. You are keeping a
↪  running note of what HAPPENS, not of what is in the room.

Write a line only when something CHANGED between the two frames and
↪  that change bears on the question: someone starts or finishes an
↪  action, an object moves, appears or is put down, text appears on
↪  screen. A thing that was already there and is still there is not a
↪  change and must not be written again. Otherwise write exactly this
↪  line and nothing else:
- nothing new

Never write a line that repeats, restates or rephrases a line already
↪  in your note. If you find yourself about to write something you
↪  have already written, write "- nothing new" instead. Repeating a
↪  line makes the note useless: it is the record of what changed, and
↪  a change written twice reads as two events. One short line per
↪  observation, each starting with "- ". Plain text only. No headings,
↪  no numbering, no bold, no JSON, no code blocks.
Do not address anyone and do not comment on your own writing.
```

Observation prompt accompanies each pair of video frames and specifies their temporal order.

AsyncLLM ProactiveVideoQA observation prompt

```
Two frames from the same camera: the first is from a moment ago, the
↪  second is now.
```

Event probe prompt is used by the event probe to determine whether the current observation provides enough information to answer the viewer's question.

AsyncLLM ProactiveVideoQA event probe prompt

```
You are watching a video in real time in order to answer this question:
↪  "{question}"

Right now, at this moment, do you see enough to give the viewer a
↪  useful answer to that question? Answer about this moment, not about
↪  whether you have answered before. Answer with one word, yes or no.
```

The speaker uses the following prompt to answer the viewer's question in one sentence based on what is happening at the current moment.

AsyncLLM ProactiveVideoQA speaker prompt

```
</think>
The viewer asked: {question}

Say what is happening RIGHT NOW that answers it. This moment only — not
↪  what happened earlier, not a summary of everything so far, not a
↪  list of steps.
```

```
One plain sentence naming the action you can see at this instant. No
↪  preamble, no mention of frames, video or watching.
Answer:
```

Mage-VL uses this prompt to generate a concise answer to the viewer's question from the current video segment.

Mage-VL ProactiveVideoQA

```
Answer this question about the video you are watching.

Question: {question}

Reply with one or two plain sentences — the answer itself and nothing
↪  else. No preamble, no mention of frames, video or watching. If you
↪  have seen only part of what the question asks about, answer with
↪  the part you have seen.
```

We use the original ProactiveVideoQA evaluation prompts to assess answer correctness. The judge scores how well the accumulated predicted answers cover the key information in the ground-truth answer.

ProactiveVideoQA judge system prompt

```
You are an evaluator for a video question answering system. Your task
↪  is to rate the whether the predicted answer covers the key points
↪  of the ground truth answer. Use the following scale to assign a
↪  score:

- 3: Mostly covered; the predicted answer covers all key information in
↪  the ground truth answer, though it may have minor inaccuracies or
↪  rephrases.

- 2: Partially covered; the predicted answer has some correct
↪  information, but also contains significant inaccuracies or missing
↪  key points.

- 1: Incorrect; the predicted answer may be related to the ground truth
↪  answer, but most of the information is missing, or the predicted
↪  answer is not relevant to the question or in very poor
↪  quality.Output the score only, do not add more explanations.
```

ProactiveVideoQA judge user prompt

```
Question: {question}

Ground Truth Answer: {ground_truth_answer}

Predicted Answer: {accumulated_predicted_answers}
```

J SELF-DEFINING ASYNCLLM AGENTS

The fact that LLM inference can now be defined by the `AsyncLLM` framework directly in Python lets us hand an agent its own runtime and have it define its own inference structure. We design a minimal harness around `Qwen/Qwen3.6-35B-A3B` that exposes mutable files containing its inference editable by the agent itself, at runtime, through tool calls. Each round, the agent's mutable `generate()` method may produce text containing tool calls that write a new inference and then patch it onto the already-running instance in place. Additionally, the agent has an asynchronous mutable method `act()` which is called every environment step. The cache blocks, background tasks, and any other states the agent has allocated survive every rewrite and can be used by both methods.

Around this loop sits an external harness: a process that repeatedly asks the agent to generate a round, applies whatever tool calls it made, scores its live environment solver against a plugged-in task environment, and reports the result back in the next round's prompt. For experimental setup described in Section 4.3 the same harness is instantiated across two ViZDoom (Kempka et al., 2016) task environments, Health Gathering and Deadly Corridor. The agent receives access to its runtime, the previous evaluation result, and the ability to modify its inference code, but no task-specific guidance on what inference structure to construct.