GOOD MEMORY HAS ECC: EVALUATING THE MEMORY OF VISION-LANGUAGE MODELS BEYOND ACCURACY

Anonymous authors
Paper under double-blind review

ABSTRACT

Memory is widely viewed as an important unsolved problem for LLMs and VLMs, and current benchmarks typically evaluate it by testing accuracy over long text or video. However, accuracy alone misses properties that matter for real long-horizon tasks. We introduce ECCBench, a benchmark and evaluation protocol that measures memory beyond a system's *capacity*—its raw accuracy at a specific budget—via three axes we call ECC: *efficiency*—the computation, in FLOPs, needed to answer from memory; *compression*—whether compressible inputs are remembered more accurately or efficiently; and *calibration*—whether the system abstains in response to its own uncertainty and the cost of an error. We find that pretrained VLMs compress their memory over text but not video and are poorly calibrated on both. Among a broader set of memory backbones, several non-Transformer architectures achieve better compression-calibration tradeoffs than RoPE Transformers, suggesting they may be useful components for agents operating over long horizons.

1 INTRODUCTION

Memory is widely viewed as an important missing capability for LLMs and VLMs, and is a subject of active research (Du et al., 2025; Long et al., 2025; Sinha et al., 2026). But what does it mean for an LLM or VLM to have a good memory and how can we evaluate it? This question may sound trivial and appears to have a settled solution already widely practiced by existing benchmarks (Packer et al., 2023; Hu et al., 2025; Wang et al., 2025). The solution is to measure accuracy of answering questions about a long text or video—if a model achieves high accuracy, it has a good memory.

But is it so? In this paper, we argue that it is not—measuring accuracy alone is insufficient and does not constitute proper evaluation of memory. We argue that additional aspects must be measured, which include Efficiency, Compression, and Calibration (ECC).

Efficiency—the computational resources needed to access memorized information—is important to measure in addition to accuracy, because without it, strong memory can be trivial. If given unlimited time, a capable but memoryless model can achieve high accuracy by simply brute-force checking every chunk of the original input, like how a person can re-read every page of a book to find the answer. Intuitively, good memory implies fast and effortless recall of information. A proper evaluation of memory must consider compute cost in addition to accuracy.

Now, if a system can accurately and efficiently answer questions about long inputs, does it necessarily have a good memory? Surprisingly not, unless the questions are chosen appropriately, because a system with poor memory can still do well by exploiting biases in the data it sees and the questions it is asked. For example, if faces occur only occasionally in a long video but every question is about them, a system that memorizes faces and nothing else beats one that remembers everything else. What if we make the questions completely random such that every piece of information will be queried with equal likelihood? In this case, given long enough input and finite storage, choosing what to remember will not help, and the only way to outperform others is *compression*—exploiting the redundancy and regularity in the input (e.g., recurring green-yellow-red patterns of a traffic light).

This suggests that proper evaluation of memory should separate compression from *saliency*—priors on what information is more important to retain. Such priors are highly task- and domain-dependent, and can in fact be counterproductive if questions are adversarially chosen. However, existing evaluations

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.

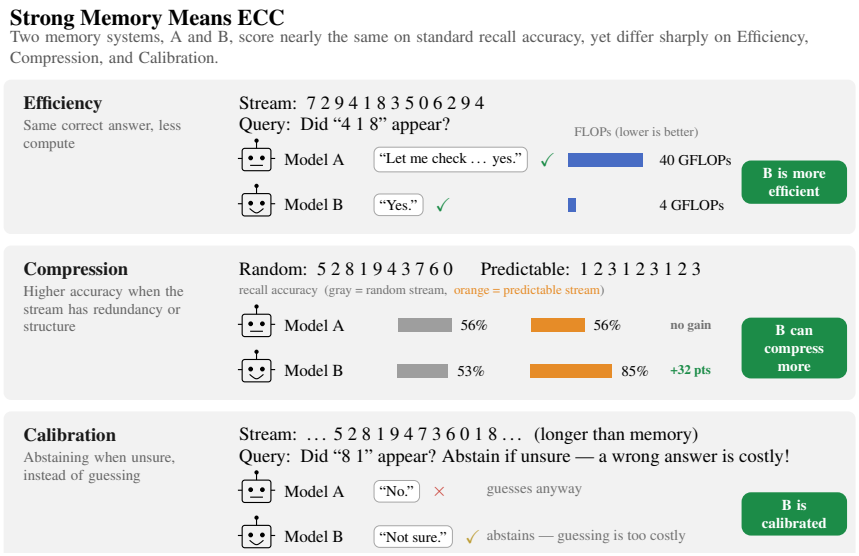


Figure 1: **Accuracy alone cannot tell two memory systems apart; ECC can.** Two black-box memory systems, A and B, answer the same recall task and score nearly the same on standard accuracy, yet differ on the axes beyond accuracy: *Efficiency* (compute spent per query), *Compression* (accuracy gained when the input has structure rather than being random), and *Calibration* (abstaining when the answer cannot be recalled and an error is costly). System B is the stronger memory on all three. Numbers are illustrative.

have not been designed to separate saliency as a confounder, and it is often unclear whether a system performs well due to saliency or compression.

Memory in real systems is also finite and cannot possibly remember everything even with compression. Thus a useful memory system needs to have *calibration*: it abstains in response to its own uncertainty and the cost of an error. A memory module should be able to abstain from responding when it detects the requested information was not preserved or is uncertain and there is a cost to incorrect retrieval. However, existing benchmarks have not been designed to evaluate calibration because in most cases the models are not provided an abstain option or a cost for an incorrect answer.

We present ECCBench, a *general* evaluation of memory that takes all of these aspects of memory into account. We measure *efficiency* through the computation required to produce an answer. Our dataset lets us measure *compression* by varying the information content of the input streams and comparing relative accuracy. Finally, we measure *calibration* by giving the system an explicit abstention option with an associated cost of error. Together, these dimensions provide a more complete picture of how effectively a system can store, retrieve, and use information. Figure 1 illustrates the three axes. In the task, a system watches a stream and is later asked whether a short candidate appeared in it. Our code is available here and the dataset here.

Our evaluation reveals significant weaknesses in VLMs. While some VLMs compress their memory on text, the ones we tested generally cannot on video. They are also poorly calibrated across both modalities; increasing the cost of an incorrect answer has little effect on their coverage.

Motivated by our VLM results, we evaluate a diverse set of memory backbones beyond the sequence lengths seen during training. We find that several architectures achieve substantially better compression-calibration tradeoffs than RoPE Transformers. Our experiments show that the underlying architecture itself can substantially change the accuracy, compression, and calibration properties of memory, and motivates developing systems with better compressive abilities.

2 Related Work

Long-context and long-video benchmarks. There is a rich corpus of work that evaluates memory by asking questions about a long document or video on both synthetic and natural data (Bai et al.,

2024; An et al., 2024; Zhang et al., 2024; Fu et al., 2025; Wu et al., 2024; Lin et al., 2024; Hsieh et al., 2024; Zhao et al., 2025; Wang et al., 2024). These test memory through accuracy but confound memory with saliency, or the ability to predict what will. Mangalam et al. (2023) formalize a related idea with the *temporal certificate*, the smallest portion of a video needed to verify an answer. While this is related to our work, the quantity we define in Section 3 captures the features of a stream that must be kept over the entire question distribution.

Definitions and evaluations of memory. Prior work uses *memory* in several senses. Early neural systems describe a state that can be read and written, while agent frameworks distinguish episodic memory for past events from semantic and procedural memory (Weston et al., 2015; Graves et al., 2014; Sumers et al., 2024), and recent work treats model components themselves as associative memories (Behrouz et al., 2025). Our definition abstracts over these mechanisms and defines a memory system as one that observes a stream, retains some state, and later answers from it. HELM is the closest precedent to our work, as it argues that models with similar task performance can behave very differently during deployment (Liang et al., 2023). It measures diverse metrics, including efficiency, calibration, and bias, but focuses on language modeling tasks and neglects compression, which is an essential characteristic of memory. MemBench and LongMemEval also evaluate memory beyond accuracy, but their tests are text-based and combine recall with conversational reasoning (Tan et al., 2025; Wu et al., 2025). In ECC, we focus on the core aspects of memory and reduce the need for semantic knowledge and multi-step reasoning, making our evaluation as general and black-box as possible across text and video.

Associative recall and compression. Associative recall is a long-standing memory test in which a cue is used to recover stored information, and remains common for comparing sequence architectures (Ramsauer et al., 2021; Graves et al., 2014; Arora et al., 2023; 2025). ECCBench uses this as the base unit of our task in different modalities because it requires little semantic knowledge or multi-step reasoning, allowing other properties of memory to be assessed independently. Compression has also been used differently in prior work, including lossless coding and methods that reduce stored activations, cache entries, or tokens, and is mainly used in relation to storage space (Delétang et al., 2023; Rae et al., 2019; Xiao et al., 2024b; Tang et al., 2024). Prior work observed that repetition improves recall in recurrent models (Arora et al., 2024), but does not generalize this to predictable structure. Our notion of compression is behavioral and can be determined empirically.

Memory architectures and agents. Many architectures aim to retain long histories using recurrent state, learned updates, or a reduced attention cache, while memory-augmented models and agents keep internal or external stores that can be searched, summarized, or edited (Katharopoulos et al., 2020; Sun et al., 2023; Gu & Dao, 2024; Dao & Gu, 2024; Yang et al., 2025a; Peng et al., 2025; Sun et al., 2025; Behrouz et al., 2024; Zhang et al., 2025; Lewis et al., 2021; Packer et al., 2023; Long et al., 2025). Rather than proposing a particular memory mechanism, ECCBench evaluates memory through additional axes, and can in principle evaluate any of these architectures.

3 BACKGROUND

We first formalize memory as an online problem over unbounded streams, and define the properties we use to evaluate it. A memory system observes an input, keeps some record of it, and later answers questions using its state. The input may arrive piecemeal and a question may be asked at any point about anything already seen. We write the stream as $x_{1:L}$, with each x_i a single observation, such as a frame or a piece of text. L is not revealed to the system in advance, and we focus on long-horizon memory and the performance as L grows. A memory system must contain a state and provide write and read operations. A write incorporates the next observation into the state and a read takes a question q and answers based on the state:

$$S_{t+1} = \mathsf{Write}(S_t, x_{t+1}), \qquad (\hat{y}, S_t^+) = \mathsf{Read}(S_t, q), \qquad \hat{y} \in \mathcal{Y} \cup \{\perp\}. \tag{1}$$

Here $\mathcal{Y}$ is the set of allowed answers and $\perp$ is abstention. A write is a general operation that may append, evict, or reorganize the memory state, and a read may occur at any t and return an updated state.

We treat memory as a black box, except insofar as we have access to the amount of FLOPs it uses during computation: F^{w} for reading the stream and F^{r} for answering. For instance, this corresponds

in pretrained VLMs to prefill and decode. In this paper, we use the total $F = F^{\mathrm{w}} + F^{\mathrm{r}}$, since a memory system may spend early so that answering is cheap or defer most computation until a question arrives. We make no assumption on which one is superior because this is task-dependent.

Let $\mathcal{D}$ be a distribution over input streams and $\mathcal{Q}$ a distribution over queries about them. For streams drawn from $\mathcal{D}$ and queries drawn from $\mathcal{Q}$, let C be the probability of correct retrieval from memory and R be the probability that the memory system responds. The probability of an incorrect answer is $R - C$, and the probability of abstention is $1 - R$.

Under this setting good memory has four desirable attributes. The first, capacity, is measured in accuracy; the three we add are the E, C, and C of the title.

Capacity. A system has high capacity if it can accurately recall a large amount of incompressible input. For a stream length L and FLOPs budget F, we measure capacity as the correct answer rate C on a matched high-entropy distribution $\mathcal{D}_{\text{high}}$. Comparing C across stream lengths at fixed F shows how much incompressible input the system can remember. The distribution $\mathcal{D}_{\text{high}}$ provides no useful repetition or other shortcut; in our synthetic examples, we define it as a uniform distribution over the vocabulary. Capacity under an unlimited FLOPs budget is analogous to raw accuracy in other benchmarks.

Efficiency. A system is efficient if it reaches a given correct answer rate at low cost, where we define cost as computation. We use total FLOPs F as the cost metric and measure efficiency as the smallest F needed to reach a target C. We use FLOPs rather than state size or latency, since storage (off of specialized hardware such as GPUs) is cheap, and latency depends heavily on hardware and implementation (Gu & Dao, 2024; Dao, 2023). All of our numbers assume one query per stream. In other settings, it may be more useful to use other metrics such as latency or the depth of computation, but using FLOPs is general and lets us consistently compare different architectures.

Compression. A system can compress if it uses stream regularity to improve recall or reduce computation. For example, it may learn the pattern of a regular sequence and be better at remembering events than over an incompressible sequence. At fixed L, F, and $\mathcal{Q}$, the accuracy gained over incompressible input is

$$\mathrm{Comp}(L, F; \mathcal{D}) = C_{\mathcal{D}}(L, F) - C_{\mathcal{D}_{\mathrm{high}}}(L, F), \tag{2}$$

where $C_{\mathcal{D}}(L, F)$ is the correct answer rate on streams from $\mathcal{D}$. Alternatively, it can be measured as FLOPs saved when reaching the same C, although this is usually more difficult to measure.

On our streams, we measure regularity with an empirical Lempel–Ziv-style statistic $\hat{h}_{\mathrm{LZ}}$ in bits per symbol (Kontoyiannis et al., 1998). Appendix A.1 gives the exact procedure and bounds of the entropy bands.

We distinguish compression from *saliency*, which is task specific. Saliency exploits knowledge of what questions are likely to be asked to decide what information is worth retaining. Compression instead exploits regularity in the stream to retain that information more efficiently. To make this distinction precise, consider the stream distribution $\mathcal{D}$ and question distribution $\mathcal{Q}$. For a stream x, let $\mathcal{K}_{\mathcal{D},\mathcal{Q}}(x)$ denote the information that must be retained to answer any question that $\mathcal{Q}$ may ask. Two streams x and x' in the support of $\mathcal{D}$ have the same $\mathcal{K}_{\mathcal{D},\mathcal{Q}}(x)$ exactly when

$$y(x, q) = y(x', q) \qquad \text{for every question } q \text{ that } \mathcal{Q} \text{ may ask,}$$

where $y(x, q)$ is the correct answer to q on x. Thus, $\mathcal{K}_{\mathcal{D},\mathcal{Q}}(x)$ preserves exactly the differences between streams that can affect a future answer.

This definition lets us disentangle saliency from compression capability. If $\mathcal{Q}$ is narrow, many differences between streams never matter, and a system can discard them using a saliency prior. On the other hand, if some questions are more likely it can further favor the information those questions require. Compression instead exploits regularity in $\mathcal{D}$ to reduce how much must be stored to preserve the distinctions that do matter. On natural tasks, $\mathcal{D}$, $\mathcal{Q}$, and therefore $\mathcal{K}$ are generally unknown. Our synthetic task controls them explicitly, together with the entropy of $\mathcal{D}$, allowing us to isolate compression from saliency.

Calibration. A system is calibrated if it answers when its memory is reliable and abstains when the risk of error is high. We define calibration here in a selective sense, and do not attempt to elicit a system's stated confidence. Because the value of abstention depends on the cost of an error, we measure calibration through coverage, R, and accuracy-on-answered, C/R, at different reward and cost profiles (El-Yaniv & Wiener, 2010).

Calibration is measured through responsiveness and selectivity. If we define R_λ as coverage when a wrong answer costs λ, answering is worthwhile when the system's confidence exceeds $\tau = \lambda/(1+\lambda)$, so raising λ should lower coverage. *Responsiveness* is the drop in coverage, $R_{\lambda_{\text{lo}}} - R_{\lambda_{\text{hi}}}$; a system that ignores the cost of an error scores zero on it. *Selectivity* asks whether the system knows which questions to abstain from by comparing its accuracy on the dropped questions against its accuracy on the kept ones.

4 Experiments

ECCBench has two evaluation sets: a synthetic set, composed of text and video subsets with controlled entropy, and a set of natural videos. We evaluate nine pretrained VLM families on the text and video versions of ECCBench under the compute-aware protocol of Section 3, using token counts to compute the FLOPs each model spent on the answer. We additionally score a RAG-like pipeline; full details and additional methods we test are in Appendix D.

4.1 Data

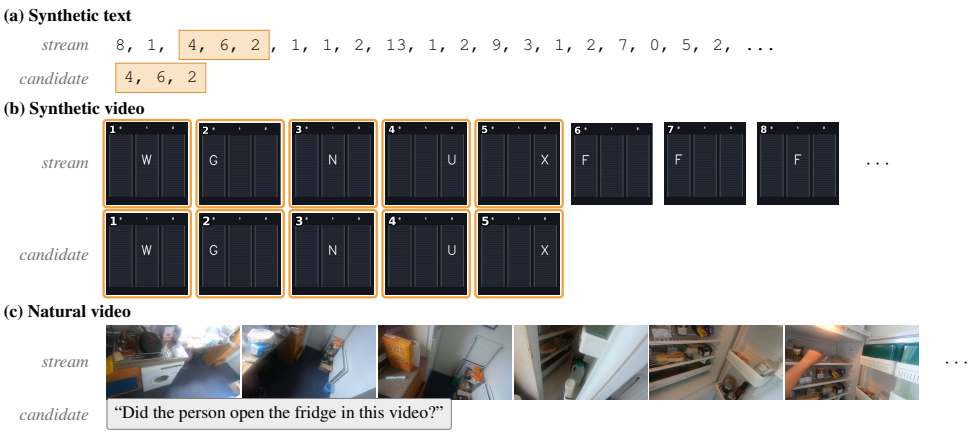


Figure 2: **One example from each evaluation set.** Each set pairs a stream with a candidate, and the model judges whether the candidate occurred in the stream. On both synthetic sets the candidate is in the same modality as the stream—text against text in (a), rendered frames against rendered frames in (b)—so no cross-modal transfer is required; highlighting marks the positions matched. The natural set (c) cannot be built that way so the stream is video and the candidate a text question. Frames in (c) are EPIC-Kitchens-100 (CC BY-NC 4.0), from our natural-video split.

Synthetic subset. Our primary evaluation uses discrete streams $x_{1:L}$ over a vocabulary $\mathcal{V}$, with $|\mathcal{V}| = 16$. For the high-entropy distribution $\mathcal{D}_{\text{high}}$, symbols are sampled i.i.d. and uniformly from $\mathcal{V}$. We generate streams using variable-order n-gram processes. The query distribution $\mathcal{Q}$ samples balanced positive and negative candidates; positives are copied from random stream positions and negatives are checked to be absent. Each stream has 16 queries asking whether a candidate occurred as an exact contiguous subsequence. Because of our construction, $\mathcal{K}_{\mathcal{D},\mathcal{Q}}(x)$ is known precisely: the questions range over all candidates of lengths three to six, and the answers depend only on which of them occur in the stream, so that is all a system needs to retain, and how often one occurs does not matter. The full generator details are in Appendix A.1.

Text streams are comma-separated integers and video streams render one letter per frame on one of three labeled lanes. The candidates are the same modality as the overall stream. Rendering and query details are in Appendix A.3. Figure 2 shows one example from each set.

Natural video subset. We build $\mathcal{D}_{\text{nat}}$ from EPIC-Kitchens-100 and SoccerNet clips and take $\mathcal{Q}_{\text{nat}}$ from activity annotations, asking whether an activity occurred in a clip. It is important to note that we can make only qualitative claims about compression on natural video because for natural video we do not know $\mathcal{D}$ or $\mathcal{Q}$. Additionally, an apparent advantage on low-density video can come from compression or from a useful saliency prior. We estimate information content through video encoding and reconstruction, then split clips at a fixed cutoff into low- and high-density bands (full details in Appendices B.6 and B.6.2). Though qualitative, this tests whether our synthetic results extend to real-world data.

4.2 Can VLMs compress?

We first ask whether VLMs can compress their memory, and if their ability to do so changes over text and video. On each of our subsets, we measure *capacity* on the densest, or highest entropy, streams and *compression* relative to the lower entropy ones. Both are read against efficiency, or FLOPs. Abstentions count as incorrect unless stated otherwise. Full details of the evaluations are in Appendix B.

We do not evaluate any closed-source models on these splits because we view efficiency as a key part of memory and we cannot calculate how much computation they use. However, we do evaluate them on an even more compressible subset and replicate our main findings in Appendix A.2.

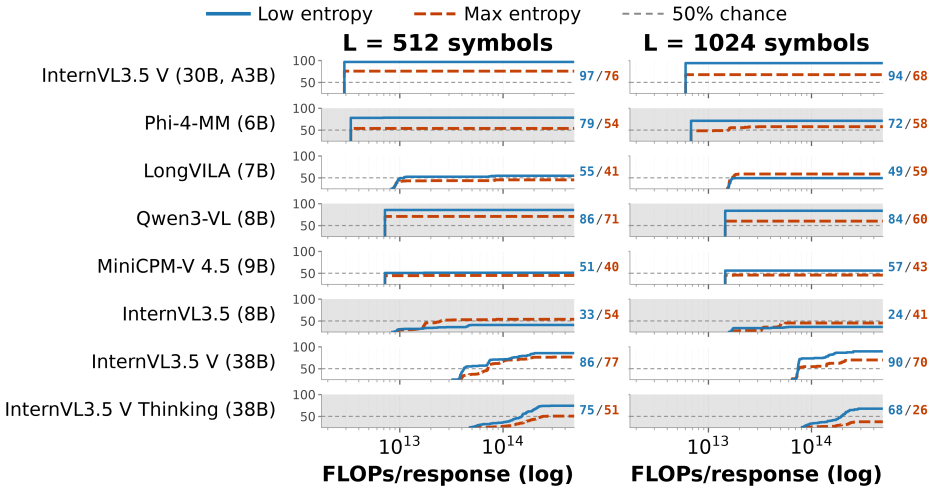


Figure 3: **On synthetic text, several models compress their memories and a few have high capacity.** Accuracy vs. FLOPs at L=512 (left) and L=1024 (right) symbols. Blue solid: low-entropy; orange dashed: high-entropy. The compression values can be read as the gap above the orange line and below the blue line at a given FLOPs value; when the orange line is above, this represents negative compression. Right-edge integers: final accuracy per condition.

4.2.1 Text

VLMs have low capacity on text, and compress their memories. The results of our experiments on text are in Figure 3. On low-entropy text, some VLMs are highly accurate. However, their performance drops precipitously on the high-entropy sequences, with only three VLMs scoring above chance, which we define throughout as at least 60% accuracy against a 50% chance rate (Appendix B.4). We also observe that models with high capacity can generally compress on the text experiments.

VLMs compress by increasing accuracy, not efficiency. There are two ways to take advantage of regularity: higher accuracy for the same amount of computation, or the same accuracy for less compute. Of the three models above chance on both entropy sets, only InternVL3.5 (38B) reaches similar accuracy at fewer FLOPs on the low entropy set. Other models fail to match their 60%+ accuracy on regular data when evaluated on irregular data, regardless of the amount of compute tested.

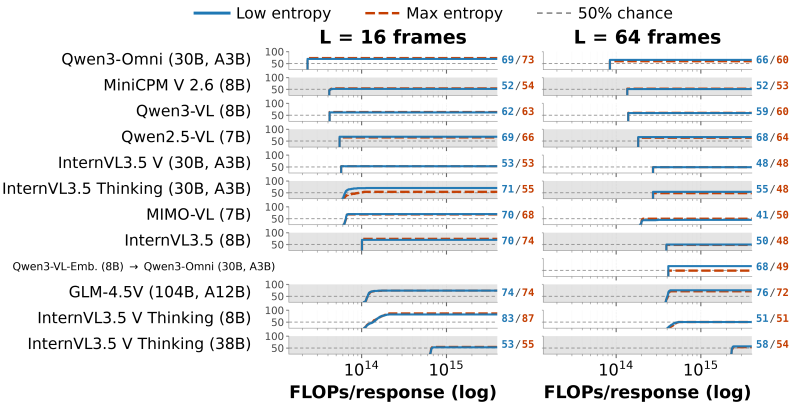


Figure 4: **On synthetic video, accuracy falls from 16 to 64 frames; at 64 frames only GLM-4.5V stays above 70%, while most VLMs fall near chance and gain nothing from predictable structure.** Accuracy vs. FLOPs at L=16 (left) and L=64 (right) frames. Blue solid: low-entropy; orange dashed: high-entropy. Right-edge integers: final accuracy per condition. Models with low accuracy on a short number of frames are not plotted to ensure we only test models with stronger perception (Appendix B.5); complete numbers are in Appendix B.4.

4.2.2 VIDEO

VLMs do not compress their memory on synthetic video. Our results for synthetic video are in Figure 4. Most models perform only as accurately as chance at both entropy levels, and the models that are performant gain little from structure. Only one VLM, InternVL3.5 Thinking (30B, A3B), has a gap larger than ten points, although the retrieval-based system we tested (Appendix D.1) compresses at $L = 64$. Capacity is also much lower on average as compared to text.

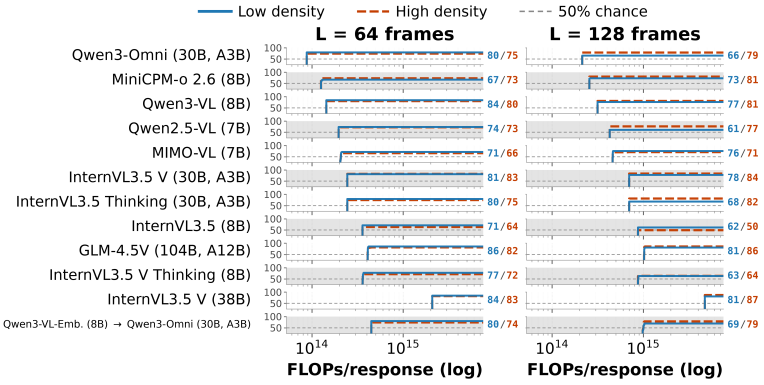


Figure 5: **On natural video, accuracy is higher than synthetic video, but gains from regularity are inconsistent.** Accuracy against FLOPs per response, one row per model and one column per clip length. Blue solid: low density; orange dashed: high density. Each curve is the running fraction of the cell answered correctly as its responses are spent cheapest first, so a wrong answer costs compute without changing the curve. The right-hand value is the cell's accuracy at maximum compute. FLOPs follow Appendix B.3.

There is little evidence of natural-video compression. Our results for natural video are in Figure 5. InternVL3.5 (8B) shows the largest low-density advantage at both lengths, gaining 7.9 and 12.4 points. This differs from InternVL3.5 (38B), which compresses text. The complete per-model contrasts are in Appendix Table 9. Many VLMs, in fact, show anti-compression, or a higher accuracy on dense videos than on sparser ones.

4.3 ARE VLMS CALIBRATED?

Next, we ask if VLMs are calibrated and test this by repeating the prior experiments with an additional prompt that adds an explicit error cost. We define Arm A, which has a payoff scheme of $+3$ for a correct answer and -1 for an incorrect answer, and Arm B, which has a payoff of $+1/-4$. The exact prompts used in these experiments are in Appendix B.7.

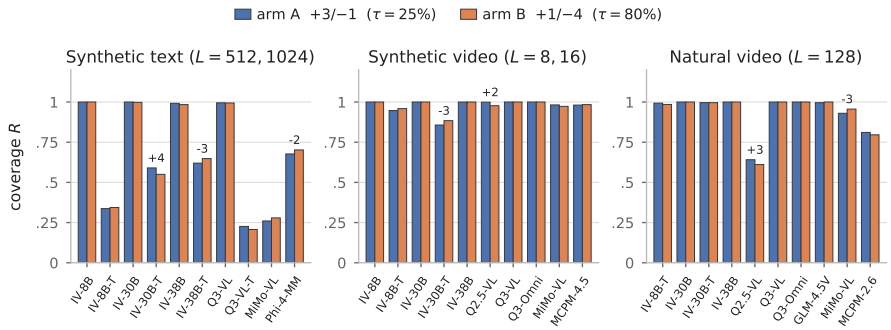


Figure 6: **VLMs are poorly calibrated on all modalities.** Coverage R under each arm, per model, for all three subsets. Nearly all models are unaffected by the increased cost of an incorrect answer, and no model abstains meaningfully more often.

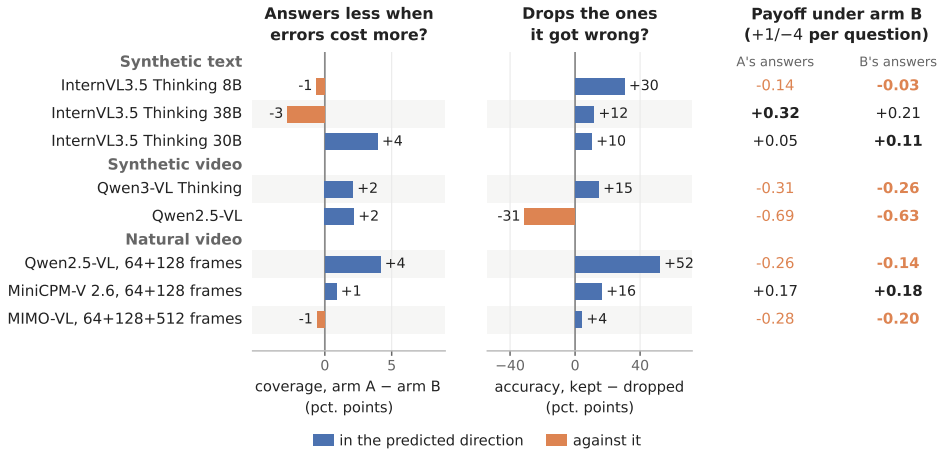


Figure 7: **VLMs only slightly optimize their payoffs.** Each model is asked to answer given a payoff scheme of $+3/-1$ (arm A) or $+1/-4$ (arm B). *Left*: does the model abstain more when a wrong answer costs four times as much? *Middle*: were the questions it abstained from on arm B the ones it got wrong on arm A? *Right*: average payoff per question with both arms scored under arm B's payoff. The arm prompts are in Appendix B.7.

VLMs respond weakly to error costs, despite some selectivity. Increasing the penalty for incorrect answers barely changes coverage (Figure 6), and many models abstain more under the more favorable payoff scheme. Nevertheless, when both sets of responses are scored under arm B, most models obtain a slightly higher payoff when prompted with that scheme (Figure 7, right). The questions they drop are generally ones they answered incorrectly under arm A (middle), indicating some selectivity. These gains remain small, and many models earn less than they would by abstaining on every question.

4.4 DO OTHER BACKBONES HAVE BETTER MEMORY THAN ROPE TRANSFORMERS?

Our prior results suggest that VLMs lack several properties required for strong memory, especially on video, motivating us to test whether architecture is one source of this limitation. We therefore

use a token-level version of ECCBench with the same task structure and vocabulary, training small models only on high-entropy streams of length $L \leq 1024$ and evaluating their ability to generalize to longer, unseen streams ($L \in \{2048, 4096\}$) at both high and low entropy. Although this is not a direct comparison with large pretrained VLMs, it isolates the memory properties of attention-based, linear-recurrent, state-space, memory-augmented, and recurrent backbones (Appendix Table 11; Appendix C). We measure confidence using the output softmax score and analyze coverage–accuracy tradeoffs, with $L = 2048$ shown in Figure 8 and $L = 4096$ in Appendix Figure 10.

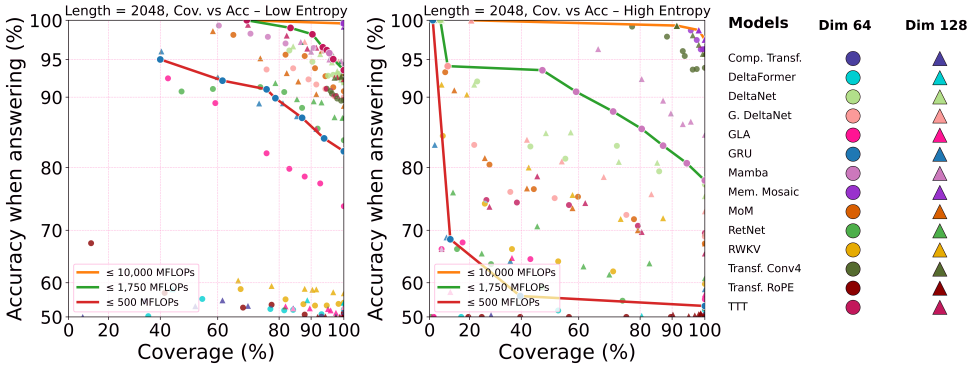


Figure 8: **Non-Transformers generally show better properties than RoPE Transformers.** Accuracy on answered queries versus coverage at ($L = 2048$), after training on ($L \leq 1024$). Left: low entropy; right: high entropy. Higher curves indicate more accurate recall at the same coverage; differences between panels show benefits from structured data. Circles and triangles denote model widths (D=64) and (128), respectively.

Several models compress despite training only on incompressible noise. Most surprising is that many models dynamically develop the ability to compress despite being trained only on incompressible noise. We can see this for individual models with high compressibility such as RetNet and TTT-MLP (Figure 8, $D = 64$). We note that due to our definition of accuracy, near-perfect accuracy can make compression difficult to measure.

Non-Transformer backbones compress better than the RoPE Transformer. We can read the two parts of Figure 8 together to compare accuracy–coverage frontiers on low and high entropy inputs. Notably, RoPE Transformers have extremely low capacity, in line with the poor length extrapolation of rotary encoding (Kazemnejad et al., 2023). Although replacing this positional encoding with a local convolution vastly increases capacity, their compression is outmatched by many models; DeltaNet, Test-Time Training Layers, and Mamba, among others, have pronouncedly higher accuracy on the left graph.

Non-Transformer models are better calibrated than the Transformer. On both graphs in Figure 8, RoPE Transformers lie near the bottom, and their coverage must be increased drastically before accuracy increases above fifty percent. Although the Transformer with the convolution has a higher accuracy-coverage curve, it spans a narrow coverage range, so lowering coverage increases accuracy very little. In scenarios where incorrect answers are extremely expensive, it is sometimes desirable to abstain on nearly all queries except ones in which the memory contains a confident answer. Lowering coverage buys very little for the Transformer as compared to models like the GRU, TTT-MLP, or GLA, for instance.

5 Conclusion

We introduced ECCBench, a framework for evaluating memory beyond accuracy. Our experiments reveal substantial weaknesses in pretrained VLMs, especially with compression and demonstrate that the backbone architecture materially affects the properties of memory. We hope our work motivates the development of new modifications to foundation models that improve these aspects of memory.

USE OF LARGE LANGUAGE MODELS

We used large language models to assist with revising and drafting manuscript text, reviewing related literature and references, writing code, running experiments and for generating figures. The authors reviewed the AI-assisted material and take responsibility for the final text, methods, results, and claims.

ETHICS STATEMENT

We have read and adhere to the Code of Ethics. All experiments were conducted without human subjects.

The goal of our work is ultimately to discover mechanisms to create better memory. While we believe this is a worthwhile goal, systems with better memory have higher potential for misuse due to their increased capabilities. For instance, better memory could enable attacks that require continuity across long interactions. Stronger memory capabilities can also support harmful applications that are only possible when acting over a long amount of time, or making many decisions. We introduce ECCBench to evaluate the properties of memory and do not claim to test the safety of the underlying model or even the safety of its memory. Responsible use of any model, especially one with strong memory, therefore requires attention to privacy, appropriate limits on retention and access, and evaluation in the intended application.

ECCBench uses synthetic streams and pre-existing video datasets. Our natural-video data is subject to their copyrights, licenses, and other restrictions. Our release on HuggingFace documents its provenance and licenses: synthetic and EasyHuman content is CC BY 4.0, EPIC-Kitchens-100-derived clips are CC BY-NC 4.0, and we provide provenance information about SoccerNet videos redistributing the restricted videos. Users who want to access and use those parts of our dataset must obtain the required access in order to download and use this data.

REPRODUCIBILITY STATEMENT

We provide an anonymous code repository and a dataset so that others can reproduce our findings. In the Appendix, we further cover details on reproducibility and further results. Appendix A describes our benchmark construction. Appendix B describes the evaluation protocols, prompts, how we counted FLOPs, and calibration experiments. Appendix C describes the backbone experiments. Together, these materials enable full reproduction of our results.

REFERENCES

Chenxin An, Shansan Gong, Ming Zhong, Xingjian Zhao, Mukai Li, Jun Zhang, Lingpeng Kong, and Xipeng Qiu. L-eval: Instituting standardized evaluation for long context language models. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pp. 14388–14411. Association for Computational Linguistics, 2024. doi: 10.18653/V1/2024.ACL-LONG.776. URL https://doi.org/10.18653/v1/2024.acl-long.776.

Simran Arora, Sabri Eyuboglu, Aman Timalsina, Isys Johnson, Michael Poli, James Zou, Atri Rudra, and Christopher Ré. Zoology: Measuring and improving recall in efficient language models, 2023. URL https://arxiv.org/abs/2312.04927.

Simran Arora, Aman Timalsina, Aaryan Singhal, Benjamin Spector, Sabri Eyuboglu, Xinyi Zhao, Ashish Rao, Atri Rudra, and Christopher Ré. Just read twice: closing the recall gap for recurrent language models, 2024. URL https://arxiv.org/abs/2407.05483.

Simran Arora, Sabri Eyuboglu, Michael Zhang, Aman Timalsina, Silas Alberti, Dylan Zinsley, James Zou, Atri Rudra, and Christopher Ré. Simple linear attention language models balance the recall-throughput tradeoff, 2025. URL https://arxiv.org/abs/2402.18668.

Yushi Bai, Xin Lv, Jiajie Zhang, Hongchang Lyu, Jiankai Tang, Zhidian Huang, Zhengxiao Du, Xiao Liu, Aohan Zeng, Lei Hou, Yuxiao Dong, Jie Tang, and Juanzi Li. Longbench: A bilingual, multitask benchmark for long context understanding, 2024. URL https://arxiv.org/abs/2308.14508.

Ali Behrouz, Peilin Zhong, and Vahab Mirrokni. Titans: Learning to memorize at test time, 2024. URL https://arxiv.org/abs/2501.00663.

Ali Behrouz, Meisam Razaviyayn, Peilin Zhong, and Vahab Mirrokni. Nested learning: The illusion of deep learning architectures, 2025. URL https://arxiv.org/abs/2512.24695.

Kyunghyun Cho, Bart van Merrienboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation, 2014. URL https://arxiv.org/abs/1406.1078.

Tri Dao. Flashattention-2: Faster attention with better parallelism and work partitioning, 2023. URL https://arxiv.org/abs/2307.08691.

Tri Dao and Albert Gu. Transformers are ssms: Generalized models and efficient algorithms through structured state space duality, 2024. URL https://arxiv.org/abs/2405.21060.

Grégoire Delétang, Anian Ruoss, Paul-Ambroise Duquenne, Elliot Catt, Tim Genewein, Christopher Mattern, Jordi Grau-Moya, Li Kevin Wenliang, Matthew Aitchison, Laurent Orseau, Marcus Hutter, and Joel Veness. Language modeling is compression. *CoRR*, abs/2309.10668, 2023. doi: 10.48550/ARXIV.2309.10668. URL https://doi.org/10.48550/arXiv.2309.10668.

Jusen Du, Weigao Sun, Disen Lan, Jiaxi Hu, and Yu Cheng. Mom: Linear sequence modeling with mixture-of-memories, 2025. URL https://arxiv.org/abs/2502.13685.

Ran El-Yaniv and Yair Wiener. On the foundations of noise-free selective classification. *J. Mach. Learn. Res.*, 11:1605–1641, 2010. doi: 10.5555/1756006.1859904. URL https://dl.acm.org/doi/10.5555/1756006.1859904.

Meadhbh I. Foster and Mark T. Keane. The role of surprise in learning: Different surprising outcomes affect memorability differentially. *Top. Cogn. Sci.*, 11(1):75–87, 2019. doi: 10.1111/TOPS.12392. URL https://doi.org/10.1111/tops.12392.

Chaoyou Fu, Yuhan Dai, Yongdong Luo, Lei Li, Shuhuai Ren, Renrui Zhang, Zihan Wang, Chenyu Zhou, Yunhang Shen, Mengdan Zhang, Peixian Chen, Yanwei Li, Shaohui Lin, Sirui Zhao, Ke Li, Tong Xu, Xiawu Zheng, Enhong Chen, Caifeng Shan, Ran He, and Xing Sun. Video-mme: The first-ever comprehensive evaluation benchmark of multi-modal llms in video analysis, 2025. URL https://arxiv.org/abs/2405.21075.

Timnit Gebru, Jamie Morgenstern, Briana Vecchione, Jennifer Wortman Vaughan, Hanna Wallach, Hal Daumé III, and Kate Crawford. Datasheets for datasets, 2021. URL https://arxiv.org/abs/1803.09010.

Alex Graves, Greg Wayne, and Ivo Danihelka. Neural turing machines, 2014. URL https://arxiv.org/abs/1410.5401.

Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces, 2024. URL https://arxiv.org/abs/2312.00752.

Cheng-Ping Hsieh, Simeng Sun, Samuel Kriman, Shantanu Acharya, Dima Rekesh, Fei Jia, Yang Zhang, and Boris Ginsburg. Ruler: What's the real context size of your long-context language models?, 2024. URL https://arxiv.org/abs/2404.06654.

Yuanzhe Hu, Yu Wang, and Julian J. McAuley. Evaluating memory in LLM agents via incremental multi-turn interactions. *CoRR*, abs/2507.05257, 2025. doi: 10.48550/ARXIV.2507.05257. URL https://doi.org/10.48550/arXiv.2507.05257.

Angelos Katharopoulos, Apoorv Vyas, Nikolaos Pappas, and François Fleuret. Transformers are rnns: Fast autoregressive transformers with linear attention, 2020. URL https://arxiv.org/abs/2006.16236.

Amirhossein Kazemnejad, Inkit Padhi, Karthikeyan Natesan Ramamurthy, Payel Das, and Siva Reddy. The impact of positional encoding on length generalization in transformers, 2023. URL https://arxiv.org/abs/2305.19466.

Chris Köcher, Alexander Kozachinskiy, Anthony Widjaja Lin, Marco Sälzer, and Georg Zetzsche. Nope: The counting power of transformers with no positional encodings. *CoRR*, abs/2505.11199, 2025. doi: 10.48550/ARXIV.2505.11199. URL https://doi.org/10.48550/arXiv.2505.11199.

Ioannis Kontoyiannis, Paul H. Algoet, Yuri M. Suhov, and Abraham J. Wyner. Nonparametric entropy estimation for stationary processesand random fields, with applications to english text. *IEEE Trans. Inf. Theory*, 44(3):1319–1327, 1998. doi: 10.1109/18.669425. URL https://doi.org/10.1109/18.669425.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive nlp tasks, 2021. URL https://arxiv.org/abs/2005.11401.

Yuhong Li, Yingbing Huang, Bowen Yang, Bharat Venkitesh, Acyr Locatelli, Hanchen Ye, Tianle Cai, Patrick Lewis, and Deming Chen. SnapKV: LLM knows what you are looking for before generation. *arXiv preprint arXiv:2404.14469*, 2024.

Percy Liang, Rishi Bommasani, Tony Lee, Dimitris Tsipras, Dilara Soylu, Michihiro Yasunaga, Yian Zhang, Deepak Narayanan, Yuhuai Wu, Ananya Kumar, Benjamin Newman, Binhang Yuan, Bobby Yan, Ce Zhang, Christian Cosgrove, Christopher D. Manning, Christopher Ré, Diana Acosta-Navas, Drew A. Hudson, Eric Zelikman, Esin Durmus, Faisal Ladhak, Frieda Rong, Hongyu Ren, Huaxiu Yao, Jue Wang, Keshav Santhanam, Laurel Orr, Lucia Zheng, Mert Yuksekgonul, Mirac Suzgun, Nathan Kim, Neel Guha, Niladri Chatterji, Omar Khattab, Peter Henderson, Qian Huang, Ryan Chi, Sang Michael Xie, Shibani Santurkar, Surya Ganguli, Tatsunori Hashimoto, Thomas Icard, Tianyi Zhang, Vishrav Chaudhary, William Wang, Xuechen Li, Yifan Mai, Yuhui Zhang, and Yuta Koreeda. Holistic evaluation of language models, 2023. URL https://arxiv.org/abs/2211.09110.

Junming Lin, Zheng Fang, Chi Chen, Zihao Wan, Fuwen Luo, Peng Li, Yang Liu, and Maosong Sun. Streamingbench: Assessing the gap for mllms to achieve streaming video understanding, 2024. URL https://arxiv.org/abs/2411.03628.

Lin Long, Yichen He, Wentao Ye, Yiyuan Pan, Yuan Lin, Hang Li, Junbo Zhao, and Wei Li. Seeing, listening, remembering, and reasoning: A multimodal agent with long-term memory, 2025. URL https://arxiv.org/abs/2508.09736.

Karttikeya Mangalam, Raiymbek Akshulakov, and Jitendra Malik. Egoschema: A diagnostic benchmark for very long-form video language understanding, 2023. URL https://arxiv.org/abs/2308.09126.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. Dinov2: Learning robust visual features without supervision, 2024. URL https://arxiv.org/abs/2304.07193.

Charles Packer, Vivian Fang, Shishir G. Patil, Kevin Lin, Sarah Wooders, and Joseph E. Gonzalez. Memgpt: Towards llms as operating systems. *CoRR*, abs/2310.08560, 2023. doi: 10.48550/ARXIV.2310.08560. URL https://doi.org/10.48550/arXiv.2310.08560.

Bo Peng, Ruichong Zhang, Daniel Goldstein, Eric Alcaide, Xingjian Du, Haowen Hou, Jiaju Lin, Jiaxing Liu, Janna Lu, William Merrill, Guangyu Song, Kaifeng Tan, Saiteja Utpala, Nathan Wilce, Johan S. Wind, Tianyi Wu, Daniel Wuttke, and Christian Zhou-Zheng. Rwkv-7 "goose" with expressive dynamic state evolution, 2025. URL https://arxiv.org/abs/2503.14456.

Samuel Planton, Timo van Kerkoerle, Leïla Abbih, Maxime Maheu, Florent Meyniel, Mariano Sigman, Liping Wang, Santiago Figueira, Sergio Romano, and Stanislas Dehaene. A theory of memory for binary sequences: Evidence for a mental compression algorithm in humans. *PLoS Comput. Biol.*, 17(1), 2021. doi: 10.1371/JOURNAL.PCBI.1008598. URL https://doi.org/10.1371/journal.pcbi.1008598.

Jack W. Rae, Anna Potapenko, Siddhant M. Jayakumar, and Timothy P. Lillicrap. Compressive transformers for long-range sequence modelling, 2019. URL https://arxiv.org/abs/1911.05507.

Hubert Ramsauer, Bernhard Schäfl, Johannes Lehner, Philipp Seidl, Michael Widrich, Lukas Gruber, Markus Holzleitner, Thomas Adler, David P. Kreil, Michael K. Kopp, Günter Klambauer, Johannes Brandstetter, and Sepp Hochreiter. Hopfield networks is all you need. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net, 2021. URL https://openreview.net/forum?id=tL89RnzIiCd.

Imanol Schlag, Kazuki Irie, and Jürgen Schmidhuber. Linear transformers are secretly fast weight programmers, 2021. URL https://arxiv.org/abs/2102.11174.

Akshit Sinha, Arvindh Arun, Shashwat Goel, Steffen Staab, and Jonas Geiping. The illusion of diminishing returns: Measuring long horizon execution in llms, 2026. URL https://arxiv.org/abs/2509.09677.

Theodore R. Sumers, Shunyu Yao, Karthik Narasimhan, and Thomas L. Griffiths. Cognitive architectures for language agents, 2024. URL https://arxiv.org/abs/2309.02427.

Yu Sun, Xinhao Li, Karan Dalal, Jiarui Xu, Arjun Vikram, Genghan Zhang, Yann Dubois, Xinlei Chen, Xiaolong Wang, Sanmi Koyejo, Tatsunori Hashimoto, and Carlos Guestrin. Learning to (learn at test time): Rnns with expressive hidden states, 2025. URL https://arxiv.org/abs/2407.04620.

Yutao Sun, Li Dong, Shaohan Huang, Shuming Ma, Yuqing Xia, Jilong Xue, Jianyong Wang, and Furu Wei. Retentive network: A successor to transformer for large language models, 2023. URL https://arxiv.org/abs/2307.08621.

Haoran Tan, Zeyu Zhang, Chen Ma, Xu Chen, Quanyu Dai, and Zhenhua Dong. Membench: Towards more comprehensive evaluation on the memory of llm-based agents, 2025. URL https://arxiv.org/abs/2506.21605.

Jiaming Tang, Yilong Zhao, Kan Zhu, Guangxuan Xiao, Baris Kasikci, and Song Han. Quest: Query-aware sparsity for efficient long-context llm inference, 2024. URL https://arxiv.org/abs/2406.10774.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need, 2023. URL https://arxiv.org/abs/1706.03762.

Weihan Wang, Zehai He, Wenyi Hong, Yean Cheng, Xiaohan Zhang, Ji Qi, Ming Ding, Xiaotao Gu, Shiyu Huang, Bin Xu, Yuxiao Dong, and Jie Tang. Lvbench: An extreme long video understanding benchmark. In *IEEE/CVF International Conference on Computer Vision, ICCV 2025, Honolulu, HI, USA, October 19-25, 2025*, pp. 22958–22967. IEEE, 2025. doi: 10.1109/ICCV51701.2025.02131. URL https://doi.org/10.1109/ICCV51701.2025.02131.

Weiyun Wang, Shuibo Zhang, Yiming Ren, Yuchen Duan, Tiantong Li, Shuo Liu, Mengkang Hu, Zhe Chen, Kaipeng Zhang, Lewei Lu, Xizhou Zhu, Ping Luo, Yu Qiao, Jifeng Dai, Wenqi Shao, and Wenhai Wang. Needle in a multimodal haystack, 2024. URL https://arxiv.org/abs/2406.07230.

Jason Weston, Sumit Chopra, and Antoine Bordes. Memory networks, 2015. URL https://arxiv.org/abs/1410.3916.

Di Wu, Hongwei Wang, Wenhao Yu, Yuwei Zhang, Kai-Wei Chang, and Dong Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory, 2025. URL https://arxiv.org/abs/2410.10813.

Haoning Wu, Dongxu Li, Bei Chen, and Junnan Li. Longvideobench: A benchmark for long-context interleaved video-language understanding, 2024. URL https://arxiv.org/abs/2407.15754.

Chaojun Xiao, Pengle Zhang, Xu Han, Guangxuan Xiao, Yankai Lin, Zhengyan Zhang, Zhiyuan Liu, and Maosong Sun. InfLLM: Training-free long-context extrapolation for LLMs with an efficient context memory. *arXiv preprint arXiv:2402.04617*, 2024a.

Guangxuan Xiao, Yuandong Tian, Beidi Chen, Song Han, and Mike Lewis. Efficient streaming language models with attention sinks, 2024b. URL https://arxiv.org/abs/2309.17453.

Mingyu Xu, Tenglong Ao, Jiaao He, Jianqiao Lu, Guang Shi, and Mingwu Zheng. Deltaformer: Unlock the state space of transformer. In Danielle Belgrave, Cheng Zhang, Laura N. Montoya, Hsuan-Tien Lin, Razvan Pascanu, Piotr Koniusz, Marzyeh Ghassemi, Nancy Chen, Iván Vladimir Meza Ruíz, and Arturo Loaiza-Bonilla (eds.), *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2025, NeurIPS 2025, San Diego, CA, USA, December 2-7, 2025 / Mexico City, Mexico, November 30 - December 5, 2025*, 2025. URL http://papers.nips.cc/paper_files/paper/2025/hash/f9f2c4cb372afceb26394e78619367b1-Abstract-Conference.html.

Songlin Yang, Bailin Wang, Yikang Shen, Rameswar Panda, and Yoon Kim. Gated linear attention transformers with hardware-efficient training, 2024. URL https://arxiv.org/abs/2312.06635.

Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving mamba2 with delta rule, 2025a. URL https://arxiv.org/abs/2412.06464.

Songlin Yang, Bailin Wang, Yu Zhang, Yikang Shen, and Yoon Kim. Parallelizing linear transformers with the delta rule over sequence length, 2025b. URL https://arxiv.org/abs/2406.06484.

Jianyu Zhang, Niklas Nolte, Ranajoy Sadhukhan, Beidi Chen, and Léon Bottou. Memory mosaics, 2025. URL https://arxiv.org/abs/2405.06394.

Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. ∞bench: Extending long context evaluation beyond 100k tokens, 2024. URL https://arxiv.org/abs/2402.13718.

Zijia Zhao, Haoyu Lu, Yuqi Huo, Yifan Du, Tongtian Yue, Longteng Guo, Bingning Wang, Weipeng Chen, and Jing Liu. Needle in a video haystack: A scalable synthetic evaluator for video mllms, 2025. URL https://arxiv.org/abs/2406.09367.

CONTENTS

1 Introduction **1**

2 Related Work **2**

3 Background **3**

4 Experiments **5**

4.1 Data 5

4.2 Can VLMs compress? 6

4.2.1 Text 6

4.2.2 Video 7

4.3 Are VLMs calibrated? 8

4.4 Do other backbones have better memory than RoPE Transformers? 8

5 Conclusion **9**

A Benchmark Details **16**

A.1 Entropy-controlled generator 16

A.2 EasyHuman 16

A.3 Video rendering and question variants 18

A.4 Dataset documentation 18

B Pretrained VLMs Experimental Details **19**

B.1 Evaluation prompts 19

B.2 Compute resources 19

B.3 Per-model FLOPs equations 20

B.4 Per-model synthetic accuracy 23

B.5 Short-L perception check 23

B.6 Natural-video evaluation 26

B.6.1 Construction 26

B.6.2 Density bands 26

B.7 Calibration: the two payoff arms 27

C Backbone Architecture Experimental Details **30**

C.1 Configurations and training 30

C.2 FLOPs formulas and per-architecture values 31

C.3 Calibration: coverage vs. accuracy 35

D Other Memory Methods **35**

D.1 RAG-Pipeline for Video Retrieval 35

D.2 Evaluating other Memory Methods for VLMs 36

E Limitations **40**

A BENCHMARK DETAILS

ECCBench is available at `https://huggingface.co/datasets/anonstreammem/substream-recollection` under component-specific terms. Synthetic and EasyHuman content is released under CC BY 4.0. EPIC-Kitchens-100-derived clips are provided under CC BY-NC 4.0. For SoccerNet, we provide only provenance information, as the underlying videos are subject to an NDA.

A.1 ENTROPY-CONTROLLED GENERATOR

Streams are drawn over a fixed alphabet $\mathcal{V}$ with $|\mathcal{V}| = 16$ symbols. Generation buckets use analytic entropy of variable-order n-gram rules (lengths 1–8). Backbone groups instead use empirical token entropy, where the low/high cutoffs are $2.0643/2.9451$ at $L = 2048$ and $0.4904/3.5938$ at $L = 4096$ (rounded). Low is at or below the first, high at or above the second.

In our synthetic experiments, we use two entropy bands that we use as examples of low- and high-density data:

- **Low entropy:** streams with strong repetition. The text streams have mean analytic entropy 0.667 bits/symbol (std 0.157), range $[0.350, 0.996]$.
- **High-entropy:** symbols sampled i.i.d. from the uniform distribution over the vocabulary; analytic entropy is $\log_2 16 = 4$ bits/symbol.

Here, a rule set is a small dictionary mapping contexts to distributions over the next item in the stream. Each context is a string of one to eight symbols from $\mathcal{V}$, meaning the generator is based on variable n-grams. For each context we specify up to four next-symbol probabilities, and the remaining probability mass spreads uniformly across the whole alphabet. For length $n > 1$, $\hat{h}_{\mathrm{LZ}} = n \log_2 n / \sum_i \lambda_i$, where λ_i is one plus the longest earlier match, allowing overlap and stopping at the stream end. To generate the next item in a stream, we find the longest suffix of the current prefix that matches some context in the rule set and then sample the next symbol from whatever the rule set defines as the output distribution. If there is no matching context, we draw uniformly over the vocabulary. To create the bands, we accept rule sets where the analytic entropy lies in the generation band.

Each (length, band) cell contains 20 sequences at all lengths that we evaluate. Each sequence is then paired with 16 questions which are a roughly equal mix of true positives and true negatives. Each candidate is three to six symbols long. A positive is a contiguous subsequence copied from a random position in the stream. A negative is sampled uniformly and is kept only if it does not occur anywhere in it. The video variant uses the same candidates rendered as frames.

Of course, repetition can help even a memory that stores random fragments, and learned rules can reject unlikely negatives without recalling exact occurrences. All models have these opportunities and may learn to exploit structure; we simply measure their benefit from regularity, because learning a rule is likely to help more than storing a recurring fragment.

Two examples of length 16, low entropy ($\hat{h}_{\mathrm{LZ}} \approx 1.02$) first and high entropy ($\log_2 |\mathcal{V}| = 4.0$) second:

```
8, 1, 4, 6, 2, 1, 1, 2, 1, 1, 2, 1, 1, 2, 1, 1
7, 1, 13, 13, 5, 2, 7, 13, 11, 9, 8, 7, 0, 0, 15, 12
```

A.2 EASYHUMAN

One of the motivations of our task is the desire to have human-like memory, which has sensitivity to sequence violations (Planton et al., 2021) and better recall for some surprising events (Foster & Keane, 2019). Each stream repeats a short pattern of length 3–5 with one or two short deviations,

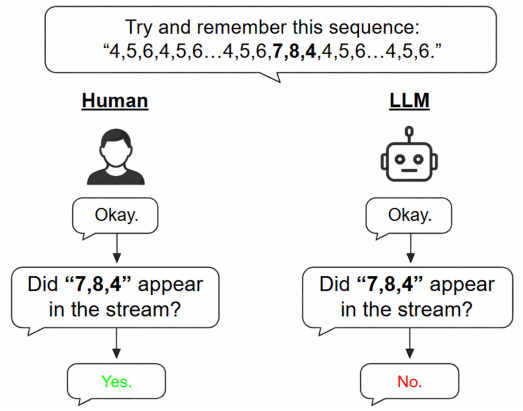


Figure 9: **Models struggle as length grows despite repeating structure.** The EasyHuman task: a short pattern repeats with one deviation. No human study was run.

and queries target the deviation and corruptions of it (Figure 9; construction below). EasyHuman is named for how obvious its redundancy is to a person, but we did not run a human study.

Models partly compress text and do not compress video at all. Results are shown in Table 1 for text and video, where L denotes the stream length (in number of comma-separated numbers for text streams, in frames for video streams). Despite the extreme redundancy of the inputs, no open-weights model reliably exceeds 77% accuracy in text, and video accuracy is near chance. For most models, text accuracy degrades only mildly from $L = 256$ to $L = 1024$, consistent with mild context rot. On video, models do very poorly and almost never abstain, which points to memory rather than perception for the models that pass the short-video check (Appendix B.5).

Frontier models also cannot compress video. Claude Opus 4.7, Gemini 3.1 Pro and GPT-5.4 Pro all score above 90 % on EasyHuman text at both lengths, which is well above any open-weight model we tested. Still, they all perform poorly on video. We show their results in Table 1 and because they are closed source and thus do not report FLOPs we do not use them in the compute-aware analysis.

Model	Text L=256	Text L=1024	Video L=256
GLM-4.5V	**76.3**	**70.5**	49.1
InternVL3.5 (8B)	72.3	66.1	47.8
InternVL3.5 Thinking (30B, A3B)	52.2	27.5	53.6
Phi-4-MM	62.5	59.4	56.2
Qwen3-VL	62.1	54.9	49.6
Qwen3-VL Thinking	11.6	9.8	53.1
MIMO-VL	41.1	37.1	52.7
API-only; no FLOPs reported			
Claude Opus 4.7	91.6	90.9	51.1
Gemini 3.1 Pro	**93.0**	**91.3**	**52.6**
GPT-5.4 Pro	**93.0**	**91.3**	51.6

Table 1: **Models can compress text, but not videos.** EasyHuman accuracy (%) for text at L=256 and L=1024 and video at L=256. Abstentions count as incorrect. No model's performance on video is distinguishable from chance, including the three closed-weights API models, which lead on text.

Construction. The EasyHuman split consists of highly regular streams with a small number of deviations, and queries that focus on the deviation and its corruptions.

We use the following structures. In all cases, the deviations are chosen such that they do not already exist in the stream.

- Repeating Pattern. A uniformly drawn motif of 3–5 symbols is repeated to fill the stream, with one or two deviations of the same length placed anywhere in it.

- Counting Disruption. The base motif is a contiguous count by 1 between two integers, either ascending (e.g., $2, 3, 4, 5$) or descending (e.g., $13, 12, 11$); motif lengths are 3–5 symbols, and the motif is repeated to fill the stream. Deviations are also 3–5 symbols long and are drawn from the uniform distribution. In video, random symbol-to-letter mapping preserves repetition but need not preserve alphabetical order.

A.3 Video rendering and question variants

Each symbol becomes one letter on one of three vertical conveyor belts (Figure 2b), rendered at 448×448 pixels and 1 frame per second, so a length-L stream is an L-frame, L-second video. The belt positions of each symbol come from a separately generated three-symbol sequence of a similar entropy level; it is structured in the low-entropy split and sampled uniformly in the high-entropy split. A letter can therefore appear on different belts within one stream.

Why glyphs and their arrangement must be retained but not other details The queries concern letter order and belt identity, not other visual details such as belt texture. Each symbol occurrence occupies one frame, in the main video and every candidate alike, with no padding or extra frames. All candidates use the same rendering style.

Glyph choice in text and video We use numbers (0-15) for text modality and letters for the video modality. We choose this because in preliminary trials that we ran, we found that VLMs read rendered letters off video frames more reliably than rendered digits at the same font size. Keeping the underlying alphabet at 16 symbols, we picked the most readable rendering for each modality, and do not believe that changing these decisions would significantly impact our results.

The synthetic video split has two question variants which are pooled in our main results. Both have candidates which are separate videos rendered using the same background, fps, and other parameters as the main video (except they are much shorter). Separate results are in Table 6:

- **Sequential.** A true positive sequential candidate is a clip taken from the original video. A false candidate may have different glyphs appear, but will never show a sequence of glyphs which did appear in the original video scrambled onto different belt positions.

- **Spatial.** A true positive, like in sequential, is in the original sequence. A negative preserves the letters of a sampled subsequence but resamples the glyphs' belt positions.

The model receives the same prompt format in both variants. The distinction lies only in how candidates are constructed and the types of incorrect answers. For both types of questions, a match must preserve symbol order and belt identity for a candidate to be a positive match.

A.4 Dataset documentation

A Croissant 1.0 `metadata.json` in the HuggingFace release is the datasheet for ECCBench and covers the standard Datasheets-for-Datasets categories (Gebru et al., 2021), including the Responsible-AI fields. The release also contains the manifest that is needed for our codebase: a top-level `questions_dataset.json` index and a per-bucket `manifest.json`. Errata and reproduction issues may be reported through the issue tracker of the code repository at `https://anonymous.4open.science/r/eccbench-anonymous-0630/README.md`, linked from the dataset card. Corrections and expanded splits will appear in the HuggingFace revision history.

B PRETRAINED VLMS EXPERIMENTAL DETAILS

B.1 EVALUATION PROMPTS

We use the following three prompts. The model can answer `{0}` for yes, `{1}` for no, or `{2}` for abstain. We parse the outputs by extracting the last bracketed token in its output text, and if no bracketed answer is emitted, the response is treated as an implicit abstention.[1]

Prompt for video questions.

```
"In the main video and option video, letters moved down three
 conveyer belts."
"A letter that appeared in any of the videos on a specific
 conveyer belt ALWAYS stayed on the same conveyer belt and
 only moved down, not left or right."
"The letters appeared sequentially. Only one was on the screen
 at a time."
"The letters continued moving down the conveyer belt until
 they left the screen."
"Examine the main video and the option video closely."
"Did this option video show events that exactly appeared
 SOMEWHERE in the main video, down to the order they appeared
 and which conveyer belt each letter was on?"
"Write {0} if yes, {1} if no, or {2} if unsure.
 Please put your answer in curly braces, e.g, {2}."
```

Prompt for text questions.

```
"You are given a main sequence of tokens and a candidate
 sequence."
"Your task: Determine if the candidate sequence appears as a
 contiguous subsequence within the main sequence, in the exact
 same order."
"The candidate must appear with all its tokens consecutive and
 in order - no gaps, no extra tokens in between."
"Answer {0} if the candidate appears as a contiguous
 subsequence, {1} if it does not, or {2} if you are unsure.
 Please put your answer in curly braces, e.g, {2}."
```

Prompt for natural-video questions.

```
Watch this video closely, paying attention to the entire video.
{question}

Write {0} if yes, {1} if no or {uncertain} if you are unsure.
Please put your answer in curly braces, e.g, {0}.
```

B.2 COMPUTE RESOURCES

All experiments ran on an internal cluster of A40, H200, L40, and RTX 3090 GPUs. The pretrained VLM evaluation jobs each allocated 2–7 GPUs per job depending on the model they evaluated. Each backbone job used a 36 wall-clock-hour training budget across four runs per architecture, on single-GPU nodes (Section C). The full project consumed approximately 60,000 GPU-hours.

[1] The video prompt below uses the spelling "conveyer" rather than the standard "conveyor"; this misspelling was present in the prompt seen by every model.

B.3 PER-MODEL FLOPS EQUATIONS

We count the FLOPs analytically. For each response we evaluate a formula whose inputs are the tokens in the query and the model's architectural constants (Table 4). Our aim in calculating FLOPs like so is a like-for-like comparison of how much arithmetic each architecture must perform to access its memory.

Constants. We obtain architectural constants from each model's checkpoint configuration on HuggingFace or its implementation if it was released elsewhere.

FLOPs calculations. A matrix multiplication of shapes $(A, B) \cdot (B, C)$ costs $2ABC$ FLOPs, two operations per multiply–add, and that is all we count. We do not count elementwise operations such as normalization, softmax, activation functions, rotary position rotations, residual additions, or embedding lookups.

Cost terms. Although there are standard, simpler ways of counting per-model FLOPs in LLMs, the vision encoder makes these more complicated. Additionally, the small differences in how these models chunk up data can make enormous differences on video data. We thus decide to use analytical formulas for FLOPs computation which take these into account. We can split the cost of a model into the vision encoder, the connector that maps vision features into the language model's input space, the language model's reading of the prompt (prefill), and its generation of the answer (decode):

$$F_{\text{total}}(L, T_{\text{in}}, T_{\text{out}}) \;=\; F_{\text{vis}} \;+\; F_{\text{conn}} \;+\; F_{\text{pre}} \;+\; F_{\text{dec}},$$

for L input frames, T_{in} prompt tokens and T_{out} generated tokens. Let d_v, f_v, and L_v denote the vision transformer's hidden dimension, FFN width, and number of blocks, respectively, with V representing the sequence length per vision block. For the language model, d_t and L_t denote its hidden dimension and layer count, n_q and n_{kv} its query and key/value head counts, and h_t its head dimension (fixed at $h_t = 128$ across all the models we evaluated), which gives query and key/value projections of dimensions $d_q = n_q h_t$ and $d_{kv} = n_{kv} h_t$. For mixture-of-experts (MoE) architectures, f_t denotes the per-expert FFN width, E the total number of experts, and k the number of active experts per token. V_{voc} is the vocabulary size, Φ represents the per-token FFN cost, and $N = T_{\text{in}} + V_{\text{LLM}}$ is the total sequence length processed by the language model, where V_{LLM} is the number of vision tokens produced by the visual connector.

Vision encoder. Each frame is cut into square patches with each patch becoming a token. We set a vision block processing V tokens, each attending over S positions, as costing

$$F_{\text{blk}}(V, S) \;=\; \underbrace{8Vd_v^2}_{\text{query, key, value, output projections}} \;+\; \underbrace{4VSd_v}_{\text{attention itself}} \;+ F_{\text{FFN}}(V),$$

$$F_{\text{FFN}}(V) = \begin{cases} 4Vd_vf_v & \text{two-matrix feed-forward,} \\ 6Vd_vf_v & \text{three-matrix (gated) feed-forward,} \end{cases}$$

and the tower also has to pay the cost of turning raw pixels into patch features, $2V_{\text{pre}}d_{\text{pat}}d_v$. Here V_{pre} is the number of patches before the class token is prepended, and d_{pat} is the number of pixel values per patch: $3p_v^2$ for patch size p_v, doubled where two frames are packed together.

The token counts below describe synthetic frames at 448×448. For our natural videos which have a resolution of 640×360, Qwen, MIMO, and GLM round each side to the nearest multiple of twice the patch size before 2×2 merging. This gives $V = 1196$ patches and 299 language tokens per two-frame group for Qwen2.5, MIMO, and GLM, or $V = 880$ and 220 for Qwen3. We use these numbers for the costs given in Table 3.

S represents how the cost of vision grows as the video gets longer:

- **Qwen2.5-VL, MIMO-VL.** Two consecutive frames are packed into one temporal group of $V = 1024$ tokens. Four layers attend to the whole group, but the remaining 28 attend only within 64-token windows ($S = 64$).
- **Qwen3-VL, Qwen3-VL Thinking, Qwen3-Omni.** All 27 blocks attend within one two-frame temporal group of $V = S = 784$ tokens.

- **GLM-4.5V.** $V = S = 1024$, one two-frame temporal group.
- **InternVL3.5.** $V = S = 1025$ per 448^2 tile: 1024 patches plus one class token.
- **MiniCPM-V 2.6 / 4.5, LongVILA, Phi-4-MM.** Each pass through the vision encoder processes a single image crop of size $V = S = 1024$ tokens. Phi-4-MM, however, explicitly divides each video frame into two crops.

Because each span is fixed, the number of spans grows proportionally with L, the frame count.

Tokens inputted to the language model. V_{LLM} follows each model's own preprocessing:

- **Qwen2.5-VL, MIMO-VL, GLM-4.5V.** Video frames are organized into temporal groups of two frames. Within each group, adjacent 2×2 patch grids are merged to one token producing 256 tokens per temporal group and $256\lceil L/2 \rceil$ visual tokens in total across all L frames.
- **Qwen3-VL, Qwen3-Omni.** Two frames per group, 2×2 merging over a 28×28 patch grid: $196\lceil L/2 \rceil$.
- **InternVL3.5.** Each video frame is processed as a single 448×448 tile (spatially). A pixel-unshuffle operation then downsamples each 2×2 grid of patch features into a single token, compressing 1,024 initial patches down to 256 tokens per frame, for a total sequence length of $256L$.
- **MiniCPM-V 2.6.** A cross-attention resampler uses 64 pre-learned query vectors to condense each frame's patch features into 64 tokens. This yields $64L$ visual tokens.
- **MiniCPM-V 4.5.** A spatio-temporal resampler processes frames in packs of size $p = \min(6, \max(1, \lceil L/180 \rceil))$. Afterwards, each pack is compressed into 64 tokens, resulting in $V_{\mathrm{LLM}} = 64\lceil L/p \rceil$ total tokens.
- **LongVILA.** Each frame is initially mapped to 256 tokens, then temporally averaged into groups of size $\mathrm{pool}(L) = 4 \cdot \mathrm{bucket}(L/256)$, where $\mathrm{bucket}(\cdot)$ rounds up to the nearest power of two in $\{1, 2, 4, 8\}$. Adding one separator token per pooled group gives $V_{\mathrm{LLM}} = 257\lfloor L/\mathrm{pool}(L) \rfloor$ total tokens.
- **Phi-4-MM.** Each frame is processed via one global view and one local crop view. Spatial pooling and inserted row-separator tokens give a cost of 545 tokens per frame ($545L$ total).

Connector FLOPs. We calculate the per-token FLOPs used by each model's connector between the vision and language towers. Let $m = 4d_v$ represent the feature dimension of a merged 2×2 patch block:

- **Qwen2.5-VL, MIMO-VL, LongVILA:** They use a patch-merging layer costing $2m^2 + 2md_t$ per token.
- **GLM-4.5V:** Follows its initial projection with a gated FFN of width 10944, yielding a total cost of $2md_t + 2d_t^2 + 6d_t \cdot 10944$.
- **Qwen3:** Uses a primary merger alongside three DeepStack adapters—which inject vision-tower features into the decoder layers—which in total cost four full merger passes at $4 \times (2m^2 + 2md_t)$.
- **InternVL3.5:** After pixel-unshuffling, it uses a two-layer linear projection costing $2md_t + 2d_t^2$.
- **Phi-4-MM:** Performs spatial projection and pooling at a cost of $2d_v d_t + 2d_t^2$.
- **MiniCPM-V (2.6 / 4.5):** Uses a cross-attention resampler. For K patch tokens mapped to 64 output tokens, the cross-attention cost is $2Kd_v d_t + 4Kd_t^2 + 256d_t K + 384d_t^2$.

The cost of prefill. We decompose the prefill FLOP cost into feed-forward/projection overhead B and causal attention A for each layer:

$$B \;=\; L_t\Big[\,2d_t(d_q + 2d_{kv}) \;+\; 2d_q d_t \;+\; \Phi\Big], \qquad A \;=\; 4L_t n_q h_t.$$

Processing a context of N tokens then requires $F_{\mathrm{pre}}(N)$ FLOPs:

$$F_{\mathrm{pre}}(N) \;=\; B\,N \;+\; A\,P(N),$$

Generating the answer (decode). Generation reuses the KV cache. Reading the prompt already produces the distribution for the first generated token, so producing T_{out} tokens applies the output vocabulary projection T_{out} times but runs the layers only $T_{\text{out}} - 1$ times, the t-th of those attending over $N + t$ cached positions:

$$\begin{aligned} F_{\text{dec}}(N, T_{\text{out}}) &= T_{\text{out}} F_{\text{voc}} + (T_{\text{out}} - 1)B \\ &\quad + A\Big[(T_{\text{out}} - 1)N + \tfrac{1}{2}T_{\text{out}}(T_{\text{out}} - 1)\Big], \\ F_{\text{voc}} &= 2d_t V_{\text{voc}}. \end{aligned}$$

with the vocabulary projection F_{voc} we count as part of decode.

Per-token feed-forward cost. For models with one gated feed-forward per layer, $\Phi = 6d_t f_t$. For mixture-of-experts models, where a small router scores the experts and only the top k run,

$$\Phi \;=\; \underbrace{2d_t E}_{\text{router}} + k \cdot 6d_t f_t \;+\; \mathbb{1}_{\text{shared}} \cdot 6d_t f_s,$$

the last term covering an always-active shared expert where the checkpoint has one (GLM-4.5V, $f_s = 1408$; Qwen3-Omni and InternVL3.5 (30B, A3B) have none). Two models are not uniform down their depth. GLM-4.5V's first decoder layer is an ordinary feed-forward of width 10944 and its remaining 45 layers are mixture-of-experts, so its body cost is $L_t\big[2d_t(d_q + 2d_{kv}) + 2d_q d_t\big] + 6d_t \cdot 10944 + (L_t - 1)\Phi$. Phi-4-MM's vision LoRA adapters ($r = 256$) are dynamically used at inference and are then added to the cost of every layer.

For synthetic video, we estimate the cost of the candidate clip using five frames, although our candidate clips vary in length. Its vision and connector costs are added separately, and its tokens are included in the context read by the language model.

The cost of a response. The prompts request a bracketed answer, such as `{0}` for yes (Section B.1). We count output through the last complete answer marker, including the marker itself, and use this length, T_{ans}, in the decoding-cost formula above.

Reading and writing costs. Recall in Section 3 we defined $F = F^{\text{w}} + F^{\text{r}}$ as the total cost of memory. Our accounting assumes one query per stream. For a VLM, which takes the whole stream into its context when the question arrives, the write cost of Section 3 includes the vision encoder, connector, and prefill of Section B.3 and the read cost is the decode.

The cost of the reading operation can be divided into two parts. We write the per-token FLOPs cost, which is independent of sequence length, as B. Attention, though, grows quadratically, and the number of pairs the model has to score is $P(N) = \frac{1}{2}N(N + 1)$. We write the FLOPs cost of a pair as A. Reading therefore costs

$$F_{\text{pre}}(N) \;=\; \underbrace{B\,N}_{\text{grows with } N} \;+\; \underbrace{A\,P(N)}_{\text{grows with } N^2}.$$

In general, reading dominates, especially on video. While there are some exceptions, notably reasoning models, many more tokens must be processed during the input than during the decode phase. Table 2 shows the three FLOPs cost variables for each VLM and Table 3 shows the writing and answering costs. Units are given in millions (MF), billions (GF), and trillions (TF) of operations.

Checkpoint context lengths. At $L = 1024$ for video, the context N exceeds the largest position the checkpoint was trained on for the three InternVL3.5 models ($6.4\times$), Phi-4-MM ($4.3\times$), MiniCPM-V 2.6 and GLM-4.5V ($2.0\times$), Qwen3-Omni ($1.5\times$), and marginally Qwen2.5-VL and MIMO-VL ($1.02\times$). The models that assign positions along separate time and space axes (the Qwen family, GLM-4.5V) advance their time index once per frame group, so their largest position index is still below the token count, whereas the InternVL and MiniCPM models index positions along a single axis and must extrapolate heavily outside of their given context length.

Model	B (MF/token)	A (kF/pair)	F_{voc} (MF)
Qwen2.5-VL (7B)	13,051	401	1,090
Qwen3-VL (8B)	13,892	590	1,245
Qwen3-Omni (30B, A3B)	5,461	786	623
GLM-4.5V (104B, A12B)	24,363	2,261	1,242
InternVL3.5 (8B)	13,892	590	1,245
InternVL3.5 (30B, A3B)	5,461	786	622
InternVL3.5 (38B)	62,411	2,097	1,556
MIMO-VL (7B)	12,759	590	1,243
MiniCPM-V 4.5 (9B)	13,892	590	1,243
MiniCPM-V 2.6 (8B)	13,051	401	1,087
LongVILA (7B)	13,051	401	1,087
Phi-4-MM (6B)	7,181	393	1,229

Table 2: The numbers we use for FLOPs accounting, where B is cost except for attention; A is what one query–key pair costs; F_{voc} is the output projection to the vocabulary distribution.

Model	synthetic text			synthetic video, 448^2			natural video, 640×360		
	N tok	F_{pre} TF	F_{dec} GF	N tok	F_{pre} TF	F_{dec} GF	N tok	F_{pre} TF	F_{dec} GF
Qwen2.5-VL (7B)	64	0.84	29	8,320	122	36	9,696	145	37
Qwen3-VL (8B)	64	0.89	32	6,400	101	39	7,168	115	40
Qwen3-Omni (30B, A3B)	64	0.35	13	6,400	51	23	7,168	59	24
GLM-4.5V (104B, A12B)	64	1.56	53	8,320	281	90	9,696	343	96
InternVL3.5 (8B)	64	0.89	32	16,512	310	51	16,512	310	51
InternVL3.5 (30B, A3B)	64	0.35	13	16,512	197	39	16,512	197	39
InternVL3.5 (38B)	64	4.00	130	16,512	1,316	199	16,512	1,316	199
MIMO-VL (7B)	64	0.82	29	8,320	127	39	9,696	151	41
MiniCPM-V 4.5 (9B)	64	0.89	32	4,224	64	36	4,224	64	36
MiniCPM-V 2.6 (8B)	64	0.84	29	4,224	59	33	4,224	59	33
LongVILA (7B)	64	0.84	29	4,240	59	33	4,240	59	33
Phi-4-MM (6B)	64	0.41	17	35,008	492	46	35,008	492	46

Table 3: What reading and writing cost, for every VLM with 64 input frames (or text symbols) and a three-token answer. N is the input length in tokens.

B.4 Per-model synthetic accuracy

The figures in Section 4.2 plot accuracy against compute, and Table 5 gives the same accuracies as numbers, including models omitted from the figures for space or for scoring below $51\,\%$ in every cell. Abstentions count as wrong.

B.5 Short-L perception check

Most models can read short streams, so their collapse at longer L is a memory failure. In order to make sure that the models have the basic skills required to perform our task, we conduct a short perception check. At $L{=}8$ little memory is needed to answer well, and the cohort of models we evaluate averages $86.8\,\%$ on text and 71.0 and $65.1\,\%$ on the video variants (Table 7). We interpret the model's $L{=}8$ score as its baseline and interpret the drop at longer lengths as degradation of its memory.

Table 4: Constants entering the per-response FLOPs formulas, read from each model's released checkpoint configuration. L_v, d_v, f_v: vision transformer depth, width, feed-forward width. $L_t, d_t, n_q/n_{kv}, f_t$: language model depth, width, query / key-value head counts, and feed-forward width (per expert where the model is mixture-of-experts). "MoE" gives the expert count E and the number active per token, with "+sh" marking an additional always-active shared expert. V_{voc} is the vocabulary size, which is used only through the output projection $F_{\text{voc}} = 2d_tV_{\text{voc}}$. "FFN style" gives the language model's / the vision tower's feed-forward form: G = two matrices, SG = three matrices (gated).

Model	L_v	d_v	f_v	L_t	d_t	n_q/n_{kv}	f_t	MoE	V_{voc}	FFN style
Qwen2.5-VL (7B)	32	1280	3420	28	3584	28/4	18944	–	152064	SG / SG
Qwen3-VL (8B) (+Thinking)	27	1152	4304	36	4096	32/8	12288	–	151936	SG / G
Qwen3-Omni (30B, A3B)	27	1152	4304	48	2048	32/4	768	128/8	152064	MoE SG / G
GLM-4.5V (104B, A12B)	24	1536	4096	46	4096	96/8	1408	128/8+sh	151552	MoE SG / SG
InternVL3.5 (8B) (+Thinking)	24	1024	4096	36	4096	32/8	12288	–	151936	SG / G
InternVL3.5 (30B, A3B) (+Think.)	24	1024	4096	48	2048	32/4	768	128/8	151936	MoE SG / G
InternVL3.5 (38B) (+Thinking)	45	3200	12800	64	5120	64/8	25600	–	151936	SG / G
MIMO-VL (7B)	32	1280	3456	36	4096	32/8	11008	–	151680	SG / SG
MiniCPM-V 4.5 (9B)	27	1152	4304	36	4096	32/8	12288	–	151748	SG / G
MiniCPM-V 2.6 (8B)	27	1152	4304	28	3584	28/4	18944	–	151666	SG / G
LongVILA (7B)	27	1152	4304	28	3584	28/4	18944	–	151651	SG / G
Phi-4-MM (6B)	27	1152	4304	32	3072	24/8	8192	–	200064	SG / G, + LoRA r=256

	text, L=512		text, L=1024		video, L=16		video, L=64	
Model	low	max	low	max	low	max	low	max
GLM-4.5V (104B, A12B)	49.2	43.4	49.1	41.0	73.8	73.8	75.6	71.5
InternVL3.5 (30B, A3B)	96.7	75.7	94.3	67.9	53.4	53.1	48.4	48.4
InternVL3.5 (38B)	85.8	76.7	89.6	70.1	–	–	–	–
InternVL3.5 (8B)	32.8	54.0	23.6	41.0	69.7	74.1	50.0	48.4
InternVL3.5 Thinking (30B, A3B)	54.1	52.6	34.0	23.9	70.6	54.7	54.5	48.1
InternVL3.5 Thinking (38B)	74.9	51.0	67.9	26.1	52.8	54.7	57.5	53.8
InternVL3.5 Thinking (8B)	25.1	24.3	14.2	15.7	82.8	86.9	50.6	50.9
LongVILA (7B)	55.2	41.1	49.1	59.0	45.6	45.6	38.8	27.2
MIMO-VL (7B)	31.1	13.9	18.9	10.4	69.7	68.4	40.6	50.3
MiniCPM-V 2.6 (8B)	15.8	22.3	14.2	16.4	52.5	54.1	51.6	53.1
MiniCPM-V 4.5 (9B)	51.4	40.1	56.6	42.5	59.1	55.3	55.3	49.7
Phi-4-MM (6B)	78.7	54.0	71.7	58.2	44.7	40.0	46.9	37.2
Qwen2.5-VL (7B)	21.9	47.5	18.9	39.6	68.8	65.9	67.5	64.4
Qwen3-Omni (30B, A3B)	23.5	35.1	25.5	26.9	69.4	72.8	65.6	60.0
Qwen3-VL (8B)	85.8	71.3	84.0	60.4	61.9	63.4	59.4	60.0
Qwen3-VL Thinking (8B)	16.9	23.8	12.3	4.5	49.4	44.7	40.9	44.1

Table 5: Per-model synthetic accuracy (%) at the lengths plotted in Figures 3 and 4, at both entropy levels. Abstentions count as wrong. InternVL3.5 (38B) was evaluated on synthetic video only at L=8 (Table 7).

	L=16				L=64			
	low entropy		max entropy		low entropy		max entropy	
Model	seq.	spatial	seq.	spatial	seq.	spatial	seq.	spatial
GLM-4.5V (104B, A12B)	73.6	73.9	74.1	73.5	73.9	77.3	78.0	65.6
InternVL3.5 (30B, A3B)	54.1	52.8	52.5	53.7	47.1	49.7	46.6	50.3
InternVL3.5 (8B)	84.9	54.7	88.0	60.5	52.2	47.9	46.6	50.3
InternVL3.5 Thinking (30B, A3B)	72.3	68.9	52.5	56.8	60.9	48.5	54.7	41.5
InternVL3.5 Thinking (38B)	52.2	53.4	58.2	51.2	59.9	55.2	55.3	52.2
InternVL3.5 Thinking (8B)	85.5	80.1	90.5	83.3	51.6	49.7	53.4	48.4
LongVILA (7B)	46.5	44.7	49.4	42.0	39.5	38.0	27.3	27.0
MIMO-VL (7B)	71.1	68.3	74.7	62.3	37.6	43.6	47.2	53.5
MiniCPM-V 2.6 (8B)	54.1	50.9	63.3	45.1	55.4	47.9	53.4	52.8
MiniCPM-V 4.5 (9B)	62.9	55.3	59.5	51.2	63.7	47.2	50.9	48.4
Phi-4-MM (6B)	48.4	41.0	43.7	36.4	43.9	49.7	34.8	39.6
Qwen2.5-VL (7B)	67.9	69.6	64.6	67.3	70.7	64.4	66.5	62.3
Qwen3-Omni (30B, A3B)	76.7	62.1	80.4	65.4	75.8	55.8	64.6	55.3
Qwen3-VL (8B)	64.2	59.6	63.9	63.0	60.5	58.3	60.2	59.7
Qwen3-VL Thinking (8B)	55.3	43.5	46.2	43.2	47.8	34.4	46.6	41.5

Table 6: Synthetic-video accuracy (%) split into sequential and spatial questions. The pooled scores weight each question equally, but the two variants have slightly different counts.

Model	Text (%) (L=8)	Video (%) (L=8, sequential)	Video (%) (L=8, spatial)
Qwen3-Omni (30B, A3B)	100.0	81.1	70.8
MiniCPM-V 4.5 (9B)	49.5	77.4	62.1
MiniCPM-V 2.6 (8B)	84.8	59.7	52.8
LongVILA (7B)	62.0	48.4	49.1
Qwen2.5-VL (7B)	16.3	66.0	67.1
Qwen3-VL (8B)	83.2	71.1	66.5
Qwen3-VL Thinking (8B)	100.0	50.3	42.2
GLM-4.5V (104B, A12B)	99.5	69.2	72.0
InternVL3.5 (8B)	100.0	91.8	49.7
InternVL3.5 Thinking (8B)	100.0	91.8	91.3
Phi-4-MM (6B)	98.9	45.3	42.2
InternVL3.5 (30B, A3B)	99.5	78.0	67.1
InternVL3.5 Thinking (30B, A3B)	100.0	86.8	84.5
MIMO-VL (7B)	96.7	75.5	73.3
InternVL3.5 (38B)	98.4	80.5	87.0
InternVL3.5 Thinking (38B)	100.0	62.9	64.0
Cohort mean	86.8	71.0	65.1

Table 7: Raw accuracy (%) at L=8 on the low-entropy band, for text and for both video question variants; abstentions count as wrong. Qwen2.5-VL's text score at L=8 comes from writing the yes and no markers swapped.

B.6 NATURAL-VIDEO EVALUATION

B.6.1 CONSTRUCTION

The natural-video benchmark consists of binary membership questions over short clips drawn primarily from EPIC-Kitchens-100, with a smaller share from SoccerNet in the same yes/no/abstain answer format as the rest of ECCBench.

Frame extraction at 1 fps. Every video is rendered as a sequence of frames which we extract at 1 fps. The frame count therefore equals the duration in seconds. Each model receives the full extracted frame sequence as input; we do not sub-sample down to a small number of frames—a common practice in long-video benchmarks where a "long" video is presented as 8, 16, or 32 uniformly sampled frames regardless of source length. Under that convention and unlike our evaluation, a model never really processes a long video.

For most videos the source clip is at least as long as the target frame count in seconds. Some source clips are shorter than the target length, and in those cases we apply a slowdown to the source clip—extracting the same content at sub-real-time speed so that 1 frame per second still yields the target frame count. We record the slowdown per-video.

Length buckets. We created six buckets, $L \in \{8, 16, 64, 128, 512, 1024\}$ frames, holding 252 videos each at $L \in \{8, 16, 64\}$, 110 each at $L \in \{128, 512\}$ and 52 at L=1024. Per-source composition: at $L \in \{8, 16, 64\}$, 236 of 252 videos per bucket are from EPIC-Kitchens-100 and 16 from SoccerNet; at $L \in \{128, 512\}$, 100 of 110 are EPIC, 10 are SoccerNet; at $L = 1024$, 42 of 52 are EPIC, 10 are SoccerNet.

Slowdown distribution. $L = 8$ and $L = 16$ have no slowed-down videos. $L = 64$ contains 2 out of 252 total. $L = 128$ contains 3 of 110. $L = 512$ contains 15 of 110. $L = 1024$ contains 42 of 52.

Event categories. Each video is associated with one of eleven event categories: `open_fridge`, `open_cupboard`, `open_drawer`, `pour_water`, `cut_nontomato`, `rinse_X`, `turn_on_tap`, `wipe_surface`, `clean_counter`, `wash_plate`, and `red_card`. The first ten are kitchen-action categories from EPIC-Kitchens-100 annotations, and `red_card` is from SoccerNet.

Question format. Each question takes the form *"Did the person [action] in this video?"* for kitchen events, or *"Did a red card appear in this video?"* for the SoccerNet category. The ground truth comes from the source annotations.

Clips and questions used in each experiment. We compare models on a subset chosen to make visual density comparable across lengths: 186, 96, 81, and 40 clips at $L = 64, 128, 512, 1024$, respectively. Each clip has a question about its associated event, plus questions about other annotated events. Figure 5 uses the same 372 questions at $L = 64$ and 366 at $L = 128$ for every method. Table 9 uses these same questions at those lengths and model-specific subsets at longer lengths. The calibration experiment uses only the additional event questions, paired across its two payoff conditions. Table 13 compares RAG and the full-video model on the same 372, 366, and 223 questions at $L = 64, 128, 512$.

Figure 5 uses MiniCPM-o 2.6. Its dimensions follow the MiniCPM-V 2.6 row in Table 4, with a vocabulary of 151,700 tokens.

B.6.2 DENSITY BANDS

Natural footage has no known ground-truth entropy, so information density is measured by fixing a coding *rate* and reading off the resulting *distortion* (Section 4.1). We use a cutoff to assign each video to the low or the high band; every question asked about a clip inherits this designation, similarly to our synthetic set.

Encoding. Each clip is re-encoded with x264 at 32 kbit per frame, at 640×360, to a constant 1 fps. Rate control is two-pass with no cap on the instantaneous rate, since single-pass average-bitrate

control does not converge on clips this short and a cap forces undershoot. The keyframe interval exceeds every clip length, so no periodic intra frame is inserted. We kept scene-cut detection enabled. The preset is `veryslow` with up to 8 B-frames and 16 reference frames.

Distortion. We compute and compare distortion in the embedding space of DINOv2 (Oquab et al., 2024). With z_t and $\hat{z}_t$ the ℓ_2-normalized patch-mean DINOv2 embeddings of the original and the decoded frame t, the per-frame distortion is

$$\delta_t \;=\; 1 - \langle z_t,\, \hat{z}_t \rangle,$$

and the clip score is the mean of δ_t over frames, dropping frame 0, which is coded from scratch.

Banding. We define a clip as low-density if its distortion score is below a cutoff of 0.0401 and consider it high-density otherwise. Table 8 shows the distortion statistics of our dataset.

L	Videos (low / high)	Questions (low / high)	Mean distortion (low / high)	Yes-rate (low / high)
64	35 / 151	70 / 302	0.0308 / 0.0690	43% / 52%
128	29 / 67	111 / 255	0.0320 / 0.0614	41% / 55%
512	29 / 52	104 / 200	0.0306 / 0.0560	46% / 51%
1024	26 / 14	98 / 48	0.0306 / 0.0489	50% / 46%
Total		383 / 805		

Table 8: Composition of the selected clip and question pool using the density cutoff of 0.0401. The yes-rate is the percentage of questions with positive ground-truth answers.

The rationale behind our metric for compressibility. The metric should agree with intuitive notions of compressibility while extending to images that differ in pixel space but carry the same information for natural language questions. For instance, duplicating frames in a video should score as more compressible on average (relative to a video of the same length without duplicated frames). The comparison in an embedding space fits these desiderata.

VLMs do not show the consistent ability to compress natural video. We show the contrast, defined as the accuracy on the low-density split minus the high-density split, in Table 9. InternVL3.5 (8B) shows the largest contrast at both lengths in Figure 5. InternVL3.5 (8B) was evaluated at $L{=}64$ and $L{=}128$ only. The two bands also differ in label prior (Table 8), so we also score every cell with both answer classes weighted equally: the mean contrast moves from $+0.55$ to $+1.60$ points, positive in 22 of 41 cells. InternVL3.5 (8B)'s gap between low and high still survives at $+12.2$ at $L{=}128$. The $95\,\%$ confidence intervals show how much our accuracy results are robust and hold up to picking a random group of videos. To estimate them, we run a cluster bootstrap where for each model and length, we build $4{,}000$ mock test sets by randomly picking clips from each density band.

Slowdown can make neighboring frames more similar, making a clip more compressible. This is a confounding factor that correlates, but perhaps is not entirely captured by, our distortion factor. We therefore repeat the comparison using only unslowed clips at $L \in \{64, 128, 512\}$. The contrast between low- and high-density accuracy is 0.33 points before removing slowed clips and 0.94 afterward, with a $95\,\%$ interval of $[-2.36, 4.18]$. We can thus still conclude, with or without slowed videos, that there is little evidence that VLMs can consistently compress natural video.

B.7 Calibration: the two payoff arms

Exact prompt modification. We append a new sentence to the original prompt in Section B.1 and do not otherwise change it. `{X}` is `{uncertain}` for natural video, `{2}` otherwise.

```
arm A:
Scoring: a correct answer earns +3 points, a wrong answer
costs −1 point, and {X} earns 0 points. Answer only if you
are more than 25% confident; otherwise answer {X}.
```

Model	contrast at L 64	128	512	1024	mean
GLM-4.5V (104B, A12B)	$+3.9$ $[-4.8, +12.1]$	-4.8 $[-12.8, +3.2]$	$+11.2$ $[+2.6, +19.3]$	$+8.0$ $[-6.0, +21.9]$	$+4.6$
InternVL3.5 (30B, A3B)	-1.7 $[-11.8, +7.8]$	-5.9 $[-15.3, +3.5]$	$+0.0^{\dagger}$ $[+0.0, +0.0]$	$+0.0^{\dagger}$ $[+0.0, +0.0]$	-3.8
InternVL3.5 (38B)	$+1.5$ $[-7.3, +10.2]$	-6.4 $[-14.3, +1.1]$	$+10.8$ $[-2.9, +24.3]$	$+5.3$ $[+0.0, +15.8]$	$+2.8$
InternVL3.5 (8B)	$+7.9$ $[-4.1, +19.2]$	$+12.4$ $[+0.7, +23.5]$	–	–	$+10.1$
InternVL3.5 Thinking (30B, A3B)	$+5.2$ $[-5.9, +15.1]$	-13.1 $[-21.8, -4.4]$	$+19.4$ $[-2.5, +40.8]$	–	$+3.8$
InternVL3.5 Thinking (8B)	$+5.6$ $[-5.3, +16.0]$	-0.9 $[-11.8, +10.1]$	$+0.0^{\dagger}$ $[+0.0, +0.0]$	$-2.9^{\dagger}$ $[-9.4, +0.0]$	$+2.4$
MIMO-VL (7B)	$+5.5$ $[-5.9, +16.9]$	$+4.7$ $[-4.0, +13.2]$	-3.3 $[-13.5, +6.8]$	-4.3 $[-21.7, +14.1]$	$+0.6$
MiniCPM-o 2.6 (8B)	-6.0 $[-20.8, +6.9]$	-7.8 $[-15.9, -0.2]$	-2.0 $[-18.2, +13.9]$	$+6.0$ $[-6.7, +19.1]$	-2.5
Qwen2.5-VL (7B)	$+1.4$ $[-10.3, +12.7]$	-15.6 $[-28.1, -3.7]$	-4.1 $[-14.5, +6.3]$	$+5.1$ $[-14.0, +26.3]$	-3.3
Qwen3-Omni (30B, A3B)	$+5.5$ $[-5.1, +15.4]$	-13.4 $[-23.6, -3.8]$	-1.2 $[-11.6, +9.5]$	-2.4 $[-15.6, +11.5]$	-2.9
Qwen3-VL (8B)	$+3.8$ $[-5.1, +12.4]$	-4.2 $[-12.8, +4.1]$	$+2.2$ $[-7.8, +10.9]$	-2.7 $[-14.4, +9.9]$	-0.2

Table 9: Each cell is accuracy on the low-density band minus accuracy on the high-density band in percentage points, with 95 % intervals underneath. A positive value means the model did better on the low-density videos. About half of the cells show a small gain and half show a small loss, and the mean over all 41 cells is $+0.55$, indicating very little compressive capability, though there may be some present that would require a large test. † marks cells where the model scores under 5 % on both bands; row means omit these cells. – marks cells that were not evaluated.

```
arm B:
Scoring: a correct answer earns +1 point, a wrong answer
costs -4 points, and {X} earns 0 points. Answer only if you
are more than 80% confident; otherwise answer {X}.
```

Calibration arm scoring. In Table 10, we show the runs behind Section 4.3 and additional data from other models.

Abstention. We are evaluating model behavior, and thus count a lack of an answer as an abstention, but three distinct behaviors are counted as abstention.

(i) Output limit. A response that reaches the 8,192 token limit without an answer counts as an abstention.

(ii) Format non-compliance or no answer. If a model does not give an answer in the proper bracketed format, or gives no answer at all, we count it as an abstention.

(iii) Explicit abstention. The model may abstain explicitly by outputting the format corresponding to that option.

Model	L	n	coverage arm A	coverage arm B	acc. answered (%) arm A	acc. answered (%) arm B	gap
synthetic text							
InternVL3.5 30B	512, 1024	644	1.000	0.998	81.8	82.3	–
InternVL3.5 38B	512, 1024	644	0.992	0.984	86.1	81.7	+0.084
InternVL3.5 8B	512, 1024	487	1.000	1.000	83.4	80.3	–
InternVL3.5 Thinking 30B	512, 1024	480	0.590	0.550	81.6	84.1	+0.102
InternVL3.5 Thinking 38B	512, 1024	644	0.620	0.648	90.5	86.6	+0.116
InternVL3.5 Thinking 8B	512, 1024	323	0.337	0.344	71.6	78.4	+0.304
MIMO-VL	512, 1024	323	0.260	0.279	60.7	63.3	+0.053
Phi-4-MM	512, 1024	644	0.677	0.702	62.6	62.8	+0.045
Qwen3-Omni	512, 1024	644	0.393	0.474	90.1	87.9	+0.178
Qwen3-VL	512, 1024	644	0.995	0.994	70.4	74.1	–
Qwen3-VL Thinking	512, 1024	323	0.226	0.207	76.7	71.6	+0.101
synthetic video							
InternVL3.5 30B	8, 16	640	1.000	1.000	48.6	48.6	–
InternVL3.5 38B	8, 16	640	1.000	1.000	48.6	48.6	–
InternVL3.5 8B	8, 16	1280	1.000	1.000	58.0	55.1	–
InternVL3.5 Thinking 30B	8, 16	1280	0.857	0.884	54.8	55.2	−0.029
InternVL3.5 Thinking 8B	8, 16	1280	0.947	0.959	53.9	54.8	+0.105
MIMO-VL	8, 16	960	0.982	0.973	72.1	73.2	+0.071
MiniCPM-V 4.5	8, 16	1280	0.981	0.984	67.5	67.9	−0.104
Qwen2.5-VL	8, 16	1280	0.999	0.977	66.2	67.1	−0.309
Qwen3-Omni	8, 16	1280	1.000	1.000	66.6	68.0	–
Qwen3-VL	8, 16	1280	1.000	1.000	72.2	69.2	–
Qwen3-VL Thinking	8, 16	1280	0.588	0.566	69.3	70.9	+0.145
natural video							
GLM-4.5V	64	186	0.995	1.000	88.6	88.2	–
InternVL3.5 30B	64	186	0.995	1.000	85.9	84.9	–
InternVL3.5 38B	64	186	1.000	1.000	89.2	89.2	–
InternVL3.5 8B	64	186	1.000	1.000	79.6	81.2	–
InternVL3.5 Thinking 30B	64	186	0.989	0.984	83.7	85.2	–
InternVL3.5 Thinking 8B	64	186	0.995	1.000	87.0	83.3	–
MIMO-VL	64	186	0.930	0.903	71.7	75.6	+0.018
MiniCPM-V 2.6	64	186	0.909	0.909	82.8	81.7	+0.441
Qwen2.5-VL	64	186	0.672	0.613	68.0	73.7	+0.476
Qwen3-Omni	64	186	1.000	1.000	80.1	79.6	–
Qwen3-VL	64	186	1.000	1.000	89.8	91.4	–
GLM-4.5V	128	270	0.996	1.000	88.5	88.5	–
InternVL3.5 30B	128	270	1.000	1.000	84.8	84.4	–
InternVL3.5 38B	128	270	1.000	1.000	93.3	93.0	–
InternVL3.5 Thinking 30B	128	270	0.996	0.996	81.4	80.3	–
InternVL3.5 Thinking 8B	128	270	0.993	0.985	82.1	76.7	–
MIMO-VL	128	270	0.930	0.956	77.3	80.2	−0.233
MiniCPM-V 2.6	128	270	0.811	0.796	84.9	86.5	+0.068
Qwen2.5-VL	128	270	0.641	0.611	75.1	76.4	+0.558
Qwen3-Omni	128	270	1.000	1.000	80.4	80.4	–
Qwen3-VL	128	270	1.000	1.000	87.4	86.7	–
GLM-4.5V	512	227	0.965	0.982	84.5	83.9	–
MIMO-VL	512	227	0.894	0.903	71.9	70.2	+0.159
Qwen3-Omni	512	227	1.000	1.000	73.6	71.4	–
Qwen3-VL	512	227	1.000	1.000	86.8	87.2	–

Table 10: All calibration data. n is the number of queries; coverage is the fraction that the model chose to answer under each arm, maxing out at 1.0. Accuracy is defined here over answered items only. *gap* is arm A accuracy on answers arm B kept minus accuracy on those that arm B abstained on, shown only where arm B declined to answer at least five questions that arm A answered.

	shared parameters				
Model	heads	FFN width	conv	Architecture-specific	D=128
Compressive Transf.	4	$4D$	—	block and memory length 128; compression $[8, 4]$ at lengths $[32, 32]$	—
DeltaFormer	4	256	—	chunked attention; qk-norm; rope_θ=10^4	—
DeltaNet	4	256	4	expand_k=expand_v=1; L2 qk-norm	—
Gated DeltaNet	4	$4D$	4	expand_v=2; gating; short-conv preprocessing	—
GLA	4	256	—	expand_k=0.5 (d_k=8); expand_v=1	—
Mamba-2	—	—	4	d_{state}=128; expand 2	d_{state}=256
Memory Mosaic	4	—	—	block size 256; 6 persistent memories of size 128	persistent size 256
MoM	4	$4D$	—	16 memories; top-k 3; capacity 1.5; Gated DeltaNet experts	—
RetNet	4	256	4	expand_k=1; expand_v=2; output gate	—
RWKV-7	4	$4D$	—	—	—
GRU	—	—	—	hidden width equals embedding width	—
Transformer (RoPE)	4	D	—	causal; rotary encoding	—
Transformer (NoPE + conv)	4	D	4	no positional encoding; depthwise conv mixer	—
TTT-MLP	4	D	—	window 16; mini-batch 256; gate	mini-batch 128

Table 11: Backbone hyperparameters, evaluated on sequence lengths 2048 and 4096, beyond the 32–1024 training window. Each architecture is three layers deep, has dropout 0, and uses a head dimension of 16. Going from D=64 to D=128 therefore doubles the width and the head count; the last column lists the settings that change beyond that. A dash means the parameter does not apply to that architecture or is not counted. FFN width is the inner width of the feed-forward layer, and D is the embedding width.

C Backbone Architecture Experimental Details

C.1 Configurations and training

Table 11 lists the architectural settings for the two embedding widths used in the backbone evaluation.

Note that we also trained a Transformer with no positional encoding at all (Kazemnejad et al., 2023; Köcher et al., 2025). It never learned the task at either width, staying at chance on the training distribution, so it is not reported. We instead use a Transformer with a small convolution to evaluate the effects of RoPE.

Training. We train three-layer versions of all architectures for up to thirty-six wall-clock hours. For each model, we take the best of four training-configuration runs and use that one for evaluation. To simulate a streaming memory backbone, where the model is prepared to answer at any time step, we train on sequence lengths 32–1024.

All jobs reuse the same synthetic membership dataset: 32 queries per sequence, 500,000 randomly generated sequences, and a vocabulary of 16 symbols. We reserve 5 % of examples for validation, train with cosine LR schedules (1 % warmup), bf16-mixed precision, and deterministic dataloading. Batch sizes range from 12 to 128 and learning rates from 2e-4 to 7e-4, chosen per architecture and width (the D=128 runs generally use 2.5e-4 and the D=64 runs 6e-4–7e-4); batch size for the largest configurations is set by GPU memory. We train for up to 125 epochs and stop once validation accuracy reaches 0.97. Per-run values for every evaluated checkpoint are released with the configurations.

C.2 FLOPs FORMULAS AND PER-ARCHITECTURE VALUES

The backbone accounting is kept separate from the VLM accounting, since these models have no vision tower and are compared only among themselves. Elementwise operations, including normalization, RoPE, activation, and softmax, are omitted under our matmul-FLOP convention.

Transformer. This derivation covers both transformer rows. Rotary positional encoding is applied elementwise and thus we do not count it under our convention, as stated before. Therefore, the RoPE Transformer and a no-encoding Transformer of the same shape have the same cost, whereas the width-4 causal depthwise convolution adds $8D$ FLOPs per token per layer, or $24DT$ over three layers. This is about $0.16\,\%$ of the total.

Each layer contains causal multi-head attention (Vaswani et al., 2023) followed by an $r{=}1$ MLP. Both use pre-normalization and residual connections. Each attention head has width $d_h = D/H$, and causal attention contains $P(T) = T(T+1)/2$ valid query–key pairs.

Q, K, V, Out projections	$4 \times (T \times D)\,(D \times D) = 8TD^2$
attention scores Q@K$^\mathsf{T}$	$2P(T)D$
weighted values A@V	$2P(T)D$
MLP, r=1	$2 \times (T \times D)\,(D \times D) = 4TD^2$
Per layer	$12TD^2 + 4P(T)D = 12TD^2 + 2DT(T+1)$

For $N = 3$ layers,

$F = 3 \cdot [12TD^2 + 2DT(T+1)]/10^6$ MFLOPs.

GRU (Cho et al., 2014). Each layer is a GRU with hidden width D. At every token, it computes input and recurrent projections for the reset gate, update gate, and candidate state.

Input projections for 3 gates	$(1 \times D)(D \times 3D)$ per token $= 6D^2$
Recurrent projections for 3 gates	$(1 \times D)(D \times 3D)$ per token $= 6D^2$
Per layer	$12TD^2$

Therefore,

$F = 12NTD^2/10^6$ MFLOPs,

with $N = 3$.

Mamba-2 (Dao & Gu, 2024). Each layer is one Mamba-2 block with:

$d_{inner} = 2D$
$d_{state} = 2D$
$p = 64$
$H = d_{inner}/p$
$d_{in} = 8D + H$
$conv_dim = 6D$

Here, $p = 64$ is the Mamba-2 SSM head width.

For recurrent inference:

input projection	$2TD \cdot d_{in}$
depthwise convolution, width 4	$8T \cdot conv_dim$
state update	$2T \cdot d_{inner} \cdot d_{state}$
state readout	$2T \cdot d_{inner} \cdot d_{state}$
output projection	$2T \cdot d_{inner} \cdot D$
Per layer	$F_{layer} = T[2D \cdot d_{in} + 8 \cdot conv_dim + 20D^2]$

For three layers,

$F = 3T[2D \cdot d_{in} + 8 \cdot conv_dim + 20D^2]/10^6$ MFLOPs.

GLA (Yang et al., 2024).

GLA. GLA uses $d_k = 8$ and $d_v = 16$ per head, alongside a rank-16 decay operation.

q and k linear layers to width $D/2$	$2D^2$
v, output gate, and final linear layers	$6D^2$
Decay layer ($D \to 16 \to D/2$)	$48D$
Memory state updates	$32D$
Feed-forward layer (width 256)	$1536D$
Per layer per token	$c = 8D^2 + 1616D$

The total compute is $F = 3cT/10^6$ MFLOPs.

RetNet (Sun et al., 2023). RetNet uses $d_k = 16$ and $d_v = 32$ per head, storing a 16×32 memory grid per head.

q and k linear layers	$4D^2$
v and output gate linear layers	$8D^2$
Final linear layer	$4D^2$
1D local convolutions (width 4)	$32D$
Memory state updates	$128D$
Feed-forward layer (width 256)	$1536D$
Per layer per token	$c = 16D^2 + 1696D$

The total compute is $F = 3cT/10^6$ MFLOPs.

DeltaNet (Schlag et al., 2021; Yang et al., 2025b). DeltaNet reads its previous state using $v_{\text{old}} = S_{t-1}^{\mathsf{T}} k_t$, calculates an update, and writes the output. Each head sets $d_k = d_v = 16$.

q, k, and v linear layers	$6D^2$
Final linear layer	$2D^2$
Learning rate projection ($D \to H$)	$D^2/8$
Three 1D local convolutions (width 4)	$24D$
Memory reading (step 1)	$32D$
Memory writing (step 2)	$32D$
Memory reading (step 3)	$32D$
Feed-forward layer (width 256)	$1536D$
Per layer per token	$c = 8.125D^2 + 1656D$

The total compute is $F = 3cT/10^6$ MFLOPs.

Gated DeltaNet (Yang et al., 2025a). Gated DeltaNet uses $d_k = 16$ and $d_v = 32$ per head with a 16×32 state grid. We omit the elementwise decay operations as they add zero matrix-multiplication FLOPs.

q and k linear layers	$4D^2$
v, output gate, and final linear layers	$12D^2$
Decay and learning rate heads	$D^2/4$
1D local convolutions (width 4)	$32D$
Memory state updates	$192D$
Feed-forward layer (width $4D$)	$24D^2$
Per layer per token	$c = 40.25D^2 + 224D$

The total compute is $F = 3cT/10^6$ MFLOPs.

RWKV-7 (Peng et al., 2025). RWKV-7 holds a 16×16 memory grid per head, using four separate matrix operations on its state per token.

State times vector pass	$2Hf^2$
State update pass	$2Hf^2$
Value-key pass	$2Hf^2$
Output readout pass	$2Hf^2$
Recurrent total ($f = 16$)	$8Hf^2 = 128D$

Combining all layers gives:

r, k, v, and final linear layers	$8D^2$
Decay low-rank layers ($D \to 64 \to D$)	$256D$
Update low-rank layers ($D \to 64 \to D$)	$256D$
Output gate low-rank layers ($D \to 128 \to D$)	$512D$
Recurrent state passes	$128D$
Extra state connection ($D \to 16 \to D$, layers 2–3)	$64D$
Feed-forward layer ($D \to 4D \to D$)	$16D^2$
Layer 1 total	$c_1 = 24D^2 + 1152D$
Layers 2–3 total	$c_2 = 24D^2 + 1216D$
Three-layer total per token	$c_{\text{tot}} = 72D^2 + 3584D$

The total compute is $F = Tc_{\text{tot}}/10^6$ MFLOPs.

DeltaFormer (Xu et al., 2025). We calculate DeltaFormer's total operations directly across all text tokens without splitting the attention process into sub-steps.

q, k, v, and final linear layers	$8TD^2$
Learning rate projection ($D \to H$)	$TD^2/8$
Feed-forward layer (width 256)	$1536TD$
Causal attention matrix operations	$4DT^2$
Per layer	$F_{\text{layer}} = T[(65/8)D^2 + 1536D] + 4DT^2$

For three layers, the total compute is $F = 3 \cdot F_{\text{layer}}/10^6$ MFLOPs.

TTT-MLP (Sun et al., 2025). Each layer uses a local window attention mechanism, an adaptive fast-weight memory, and a standard feed-forward network. Each head stores a $16 \to 64 \to 16$ network inside its state. We denote the total count of valid attention token pairs across window length W as $P_W(T) = \sum_{t=1}^{T} \min(t, W+1)$. When $T > W$, this evaluates to $P_W(T) = (W+1)T - W(W+1)/2$. Setting $W = f = 16$, we compute:

Window attention linear layers	$8TD^2$
Window attention matrix operations	$4D \cdot P_W(T)$
Fast-weight memory linear layers	$10TD^2$
Learning rate projection ($D \to H$)	$TD^2/8$
Fast-weight memory updates	$896DT$
Feed-forward layer ($D \to D \to D$)	$4TD^2$
Per layer	$F_{\text{layer}} = T[22.125D^2 + 896D] + 4D \cdot P_W(T)$

For three layers, the total compute is $F = 3 \cdot F_{\text{layer}}/10^6$ MFLOPs.

MoM (Du et al., 2025). Each token updates four memory paths: three routed paths and one shared path. Each memory path uses a Gated DeltaNet state with $d_k = d_v = 16$. Because the feature projections are shared across branches, we compute them once per token. We maintain separate convolution and memory states across all four paths:

Routing layer ($D \to 16$)	$32D$
Shared feature linear layers	$6D^2$
Shared decay and learning rate heads	$D^2/4$
Four active memory paths:	
1D local convolutions (width 4)	$96D$
Memory state updates	$384D$
Output gate and final linear layers	$4D^2$
Feed-forward layer (width $4D$)	$24D^2$
Per layer per token	$c = 34.25D^2 + 512D$

The total compute is $F = 3cT/10^6$ MFLOPs.

Memory Mosaic (Zhang et al., 2025). When reusing cached sequence states, we omit the exponential moving average operations as they add zero matrix-multiplication FLOPs. We count the full sequence attention operations directly:

Context key, value, and output linear layers	$6TD^2$
Persistent key and output linear layers	$4TD^2$
Sequence attention matrix operations	$2DT(T-1)$
Persistent memory reads (6 reads, width $2D$)	$48TD^2$
Per layer	$F_{\text{layer}} = 58TD^2 + 2DT(T-1)$

For three layers, the total compute is $F = 3 \cdot F_{\text{layer}}/10^6$ MFLOPs.

Compressive Transformer (Rae et al., 2019). This architecture processes input sequences in blocks of 128 tokens. New tokens attend to other tokens in the same block as well as m_b past saved memory slots. For a block b, the total number of attention pairs is $P_b = 128 \cdot m_b + 8256$.

The number of active memory slots m_b follows a warm-up sequence: $m_b = 0, 128, 144, 160, 164, \ldots, 193, 193, \ldots$

Token linear layers	$1024D^2$
Feed-forward layer ($D \to 4D \to D$)	$2048D^2$
Gated residual connections	$3072D^2$
Attention matrix operations	$6D \cdot P_b$
Full block cost (before memory cleanup)	$6144D^2 + 6D \cdot P_b$

We perform periodic memory cleanup steps at specific block milestones:

Block 2: 8× compression and cached updates	$320D^2$
Block 4: 4× compression and cached updates	$48D^2$
Block 12: Long-term memory update and cached updates	$32D^2 + 16D$

Any other token attends to 193 past memory slots and costs $48D^2 + 1164D$.

For $n \geq 12$ full blocks followed by one token,

$F = 3 \cdot [\sum_b (6144D^2 + 6D \cdot P_b) + (n-1) \cdot 320D^2 + (n-3) \cdot 48D^2 + (n-11) \cdot (32D^2 + 16D) + 48D^2 + 1164D]/10^6$ MFLOPs.

Model	D=64, MFLOPs		D=128, MFLOPs	
	L=2048	L=4096	L=2048	L=4096
Compressive Transformer	1817.09	3711.26	6188.52	12550.06
DeltaFormer	4033.22	14508.49	8475.58	29835.08
DeltaNet	856.06	1711.69	2121.26	4241.48
GLA	837.17	1673.94	2077.19	4153.37
Gated DeltaNet	1101.54	2202.55	4229.92	8457.78
Mamba-2	926.87	1853.29	3669.71	7337.63
Memory Mosaic	3071.73	9363.97	9064.12	24567.84
MoM	1063.78	2127.03	3852.25	7702.62
RWKV-7	1074.27	2148.01	3357.08	6712.52
RetNet	1070.07	2139.62	2945.84	5890.24
GRU	302.14	604.13	1208.55	2416.51
TTT-MLP	936.21	1872.06	2986.54	5971.84
Transformer (RoPE)	1915.11	7051.30	4434.49	15310.85
Transformer (conv)	1918.26	7057.59	4440.79	15323.44

Table 12: Backbone model computational complexity across embedding dimensions ($T = L + 1$). These are the values used for the Pareto frontiers in Figures 8 and 10.

C.3 Calibration: coverage vs. accuracy

For a fixed MFLOPs budget, we read increased (i) capacity as shifting the whole curve up and to the right, (ii) compression as a larger low- vs. high-entropy gap at the FLOPs budget, and (iii) calibration as better tradeoffs between coverage and accuracy.

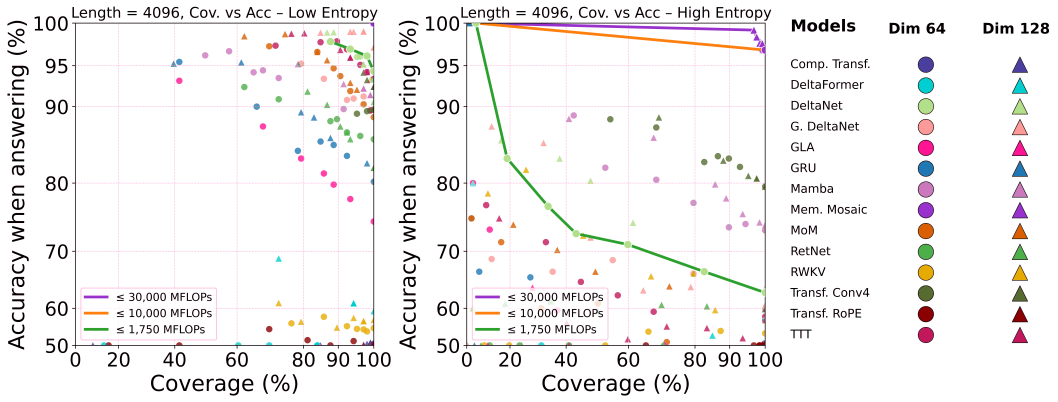


Figure 10: Same coverage-vs-accuracy convention and legend abbreviations as Figure 8, now at $L = 4096$. Both panels use FLOPs budgets $\leq$30,000, $\leq$10,000, $\leq$1,750 MFLOPs (left: low-entropy split; right: high-entropy split). The $\leq$30,000 tier admits all configurations.

D Other Memory Methods

D.1 RAG-Pipeline for Video Retrieval

Our goal for ECC is to provide a general framework that can apply to many different type of systems. We therefore also evaluate our system on a RAG-like model, with an embedding and search procedure. Each video is split into 16-frame windows at a stride of 4 and every such windowed clip is embedded with Qwen3-VL-Embedding-8B. When a query is received, the candidate clip is embedded with the same model. We then rank it against all windows, and the top-ranked window (the top three at $L \geq 128$) is retrieved. This is then sent to Qwen3-Omni (30B, A3B) with the same prompts as every baseline, plus a sentence saying that the clip was retrieved to provide some context for calibration purposes.

The embedder shares the Qwen3-VL (8B) architecture, so we reuse those constants to calculate its prefill cost. Embedding a query costs 0.56 trillion on natural video, where the query is a sentence, and 11 trillion on synthetic video, where the query is a video clip. Qwen3-Omni then answers as if on a clip of the retrieved length exactly as in Section B.3.

	accuracy (%)		one query (TF)	
corpus	RAG	full video	RAG	full video
natural L=64	75.3	75.5	447	87
natural L=128	76.0	75.1	994	213
natural L=512	73.5	70.0	4,153	1,786
synthetic L=64	58.4	62.8	413	84

Table 13: The RAG-like pipeline versus Qwen3-Omni when it receives the full video, answering the same questions. RAG costs include building the index, embedding the query, and answering from the retrieved clip.

For q questions about the same stream, a comparison that reuses both systems' preparation gives average costs per answer

$$\overline{F}_{\mathrm{RAG}}(q) = W_{\mathrm{RAG}}/q + R_{\mathrm{RAG}},$$
$$\overline{F}_{\text{cached VLM}}(q) = W_{\mathrm{VLM}}/q + R_{\mathrm{VLM}}.$$

Here W is the one-time preparation cost and R the cost of answering each question. This assumes the VLM can retain and reuse its processed stream. Spreading preparation costs over questions does not necessarily make RAG cheaper than this alternative; generally, it will be cheaper when the embedding model is smaller or more efficient than the model that directly answers the questions.

Table 13 shows three main results. On natural video, the retrieval pipeline scores within one point of the full-video baseline at $L = 64$ and 128, and higher at $L = 512$, but costs more when each answer includes building the index. On synthetic video, the pipeline scores 4.4 points below the full-video baseline and answers only $37.0\,\%$ of yes-questions correctly. We also ran a retrieval probe to see how often the RAG model retrieves the right window; it succeeded only $33\,\%$ of the time in this setting. On natural video, no-questions are harder because the event is absent: the retriever must still return its closest match, and the model rarely rejects that window as irrelevant, causing no-question accuracy to fall from $53.8\,\%$ at L=64 to $46.8\,\%$ at L=512. Intuitively, retrieval also makes calibration harder because even weak matches are presented as evidence of a match (and the embedder never abstains). We theorize that providing retrieval similarity or confidence scores, as well as other metadata about the sequence, might help the model judge the quality of a match and infer what else is contained in the source video.

D.2 Evaluating other Memory Methods for VLMs

Our evaluation is also designed to help in testing the impact of other modifications to memory systems, such as those designed to improve memory or save compute costs. We therefore use ECCBench to test recall, the benefit from regularity, and calibration at a given FLOPs budget for SnapKV (Li et al., 2024) and InfLLM (Xiao et al., 2024a), which reduce attention to past tokens without requiring retraining the model. We test Mistral-7B-Instruct-v0.2 on text questions of $L = 512$ and $L = 1024$ symbols, and InternVL3.5-8B on $L = 16$ synthetic video frames. Our full control keeps all past tokens available. We use greedy decoding and output at most $8{,}192$ tokens per question.

Using the question to choose memory. For InfLLM, we keep the context in 128-token blocks and retrieve four or eight blocks that best match the tokens it is processing, including the question. Each block is represented by its four most-attended positions per head. It also keeps 128 initial tokens for the initial prompt, and a recent window of 512 tokens for text or 2048 for video (for the candidate).

SnapKV instead selects once per question, using attention from the last 32 positions and smoothing each position's score over five neighboring positions. For text, we use ordinary SnapKV with a 1024-position cache per head, which can also remove question tokens. For video, we select 1024 stream token positions per head and retain the entire candidate and question. We likewise retain the

candidate and question throughout answering with video InfLLM, expanding its recent window when necessary.

Choosing memory before seeing the question. We also adapt SnapKV to select 1024 positions from the stream before adding the question. This discards information without knowing what will be asked and will be expected to lose considerable accuracy, especially on incompressible data. Repetition can make that loss less harmful.

Accuracy, cost, and regularity. When SnapKV sees the question first, Mistral scores $77.2\,\%$ against $76.9\,\%$ for its full-cache control while saving $7.3\,\%$ of mean FLOPs. InfLLM with four blocks scores $75.3\,\%$ against $76.7\,\%$, saving $9.7\,\%$. If we allow the model to select the appropriate tokens before the question with SnapKV-1024 it saves $9.9\,\%$ of FLOPs, but accuracy falls to $64.1\,\%$. At $L = 1024$, regular streams score 32.5 points above random streams with full memory, compared with 13.8 points with this version of SnapKV, showing worse compression ability (Figure 11). On video, SnapKV after the question scores $64.7\,\%$ against $61.7\,\%$, but costs $0.05\,\%$ more. We suspect selection helps focus attention on relevant video tokens. InfLLM with four blocks saves $2.4\,\%$ of FLOPs, but its accuracy falls significantly from $63.4\,\%$ to $53.0\,\%$.

Calibration under changing error costs. We then evaluate the calibration properties of these methods. When errors cost more (under arm B), Mistral answers more often with SnapKV-1024 selecting before the question and InfLLM using four blocks, with larger changes than the full-cache control. This does not consistently help: SnapKV earns more points and InfLLM earns fewer, but both remain below full memory and the option of always abstaining on each question. InternVL answers every question under both rules with all methods, so neither improves abstention; both earn less than always abstaining under B (Figure 12).

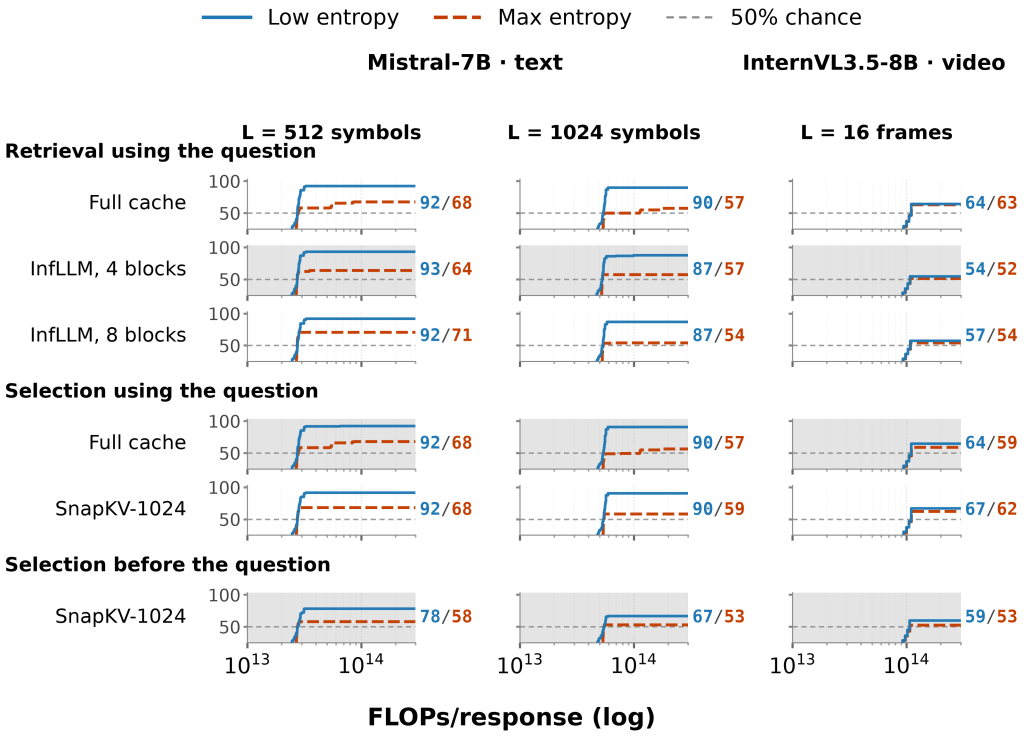


Figure 11: Corresponding figure to Figure 3. Each curve shows the percentage of all questions answered correctly within the budget including scoring and other preparation costs. Blue solid and orange dashed curves show low and maximal entropy. Numbers at the right give final accuracy.

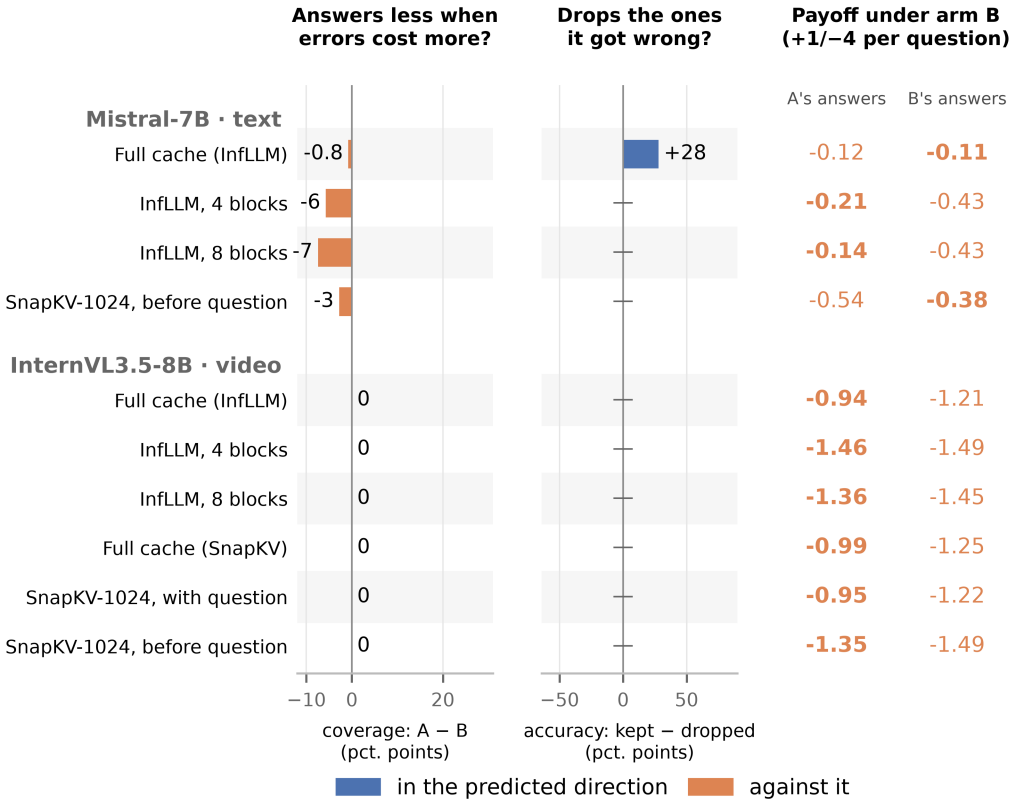


Figure 12: Calibration as in Figure 7. Left: coverage under A $(+3/-1)$ minus B $(+1/-4)$. Middle: difference in A accuracy between answers B keeps and drops; — if either group is empty or fewer than five answers are dropped. Right: points earned under B's rule, with the higher value bold.

Counting computation. We account for FLOPs for these methods following Appendix B.3. SnapKV reads the full stream before selecting positions. Selection after the question reduces attention only during generation, because prefill still reads the whole stream. InfLLM restricts attention during stream reading and answering. We include the extra attention scores used by SnapKV and the block-comparison scores used by InfLLM in the FLOPs calculations.

For D decoder layers, width d, feed-forward width f, H query heads, K key/value heads, head width h, and vocabulary size V, the per-token projections and feed-forward cost per layer is $C = 4dHh + 4dKh + 6df$. With N input tokens and T charged output tokens, the total is

$$F = F_{\text{vision}} + DC(N + T - 1) + 4h \sum_{\ell=1}^{D} A_\ell + 2dVT + S.$$

Here A_ℓ counts attention pairs across all heads during reading and generation. The input pass predicts the first output token. For video, F_{vision} follows Appendix B.3.

For SnapKV, scoring n tokens with a window of $w = 32$ adds $S = 2DHh\{w(n-w) + w(w+1)/2\}$ when selection is necessary. For InfLLM, scoring g candidate blocks costs $2Hhg$ per layer per retrieval. S is the sum of these costs. When all available blocks fit the retrieval budget, block scoring is unnecessary. For RAG, scoring M stored window vectors of length d_e costs $2Md_e$ FLOPs per question. Across all three methods, we count scoring products but exclude softmax, normalization, sorting, and transfers.

E LIMITATIONS

Limitations. Our study codifies a general test of memory, but we do not test all types of questions that cover memorization. We omit natural text data, and while we do use natural video data, we draw the bulk of our conclusions from synthetic data. ECCBench isolates associative recognition and not the full range of behaviors commonly called memory. Our goal is not to test confounding skills that often impact tasks where memory is used, such as temporal reasoning, interference between different episodes or experiences, or whether remembered information can subsequently help complex tasks. ECC is intended to apply to the memory used in these settings, but we do not test on them.

We measure efficiency using arithmetic FLOPs and assume one query per stream. FLOPs provide an implementation-independent approximation to computation, but there are many other qualities that matter like memory bandwidth and communication costs. Moreover, systems that amortize an expensive write over many subsequent queries may rank differently under a multi-query workload.

Our synthetic experiments are designed to remove other skills and priors because what must be retained is known exactly, but they still rely on the ability of the model to extract and retain that information effectively. Thus, it is possible some models are able to store certain glyphs more compactly due to their priors, which would inflate their capacity score.

Our compressibility measure for natural video is a heuristic, validated only on sanity checks such as slowed-down clips scoring as more compressible. We chose it because no established measure of compressibility in feature space, rather than pixel space, exists.

We do not report complete results for closed-source models because we cannot access how many FLOPs they use. However, our definitions and axes of memory still apply to them.

Our backbone study uses small models, which limits the ability to generalize to LLM-scale, where models have billions of parameters. At that scale, it is possible that other capabilities emerge, and that the differences we observe are more or less pronounced. Some backbones may benefit from a wider hyperparameter search than the four configurations we tried. Nonetheless, as many studies of small models do generalize to larger ones, we believe it will be useful in motivating more efficient memory.