EDGE RESEARCH · TECHNICAL REPORT 03

The VLM above the LLM: a layered architecture for live video

Perception on top, language underneath. Why we put the vision language model (VLM) first, and what the language model (LLM) is allowed to do.

Edge Research, Edge Video AI Version 1.0 · 15 Nov 2026

Status: Edge Research technical report. Not peer reviewed. It describes Edge's own systems and internal measurements; every number carries its scope. Third-party work is cited with links.

ABSTRACT

Most AI systems that handle video bolt a vision encoder onto a language model and let the language model drive. For live video we invert the emphasis. In Edge Video AI's architecture the vision language model (VLM) is the top layer: it watches the stream, follows every moment and knows when the ad break comes, and it is the only layer allowed to say what is on screen. A language model sits underneath as a support layer for reasoning, for understanding what a person asks for, and for building the apps that run on top of the VLM's signals. We describe the layers, the four rules that keep them apart, how customer context enters before perception rather than after it, how cost is controlled by reading each channel once and recognising familiar scenes locally, and the role of the Control Room as a live view of what the engine is doing. We close with limitations and open questions.

1. Introduction

Language models have become the default interface to AI. It is natural to treat video the same way: turn the picture into text, then let a language model do the rest. For recorded clips that can work. For live television it fails in two predictable ways. It is slow and expensive, because every moment must be described in words before anything else happens. And it is unreliable, because once the picture has been turned into text, the language model has no way to check its own story against what was actually on screen.

Our architecture starts from the other end. The question "what is on screen right now?" is answered by the vision language model and only by it. Everything else, including the language

model, works from the VLM's answer.

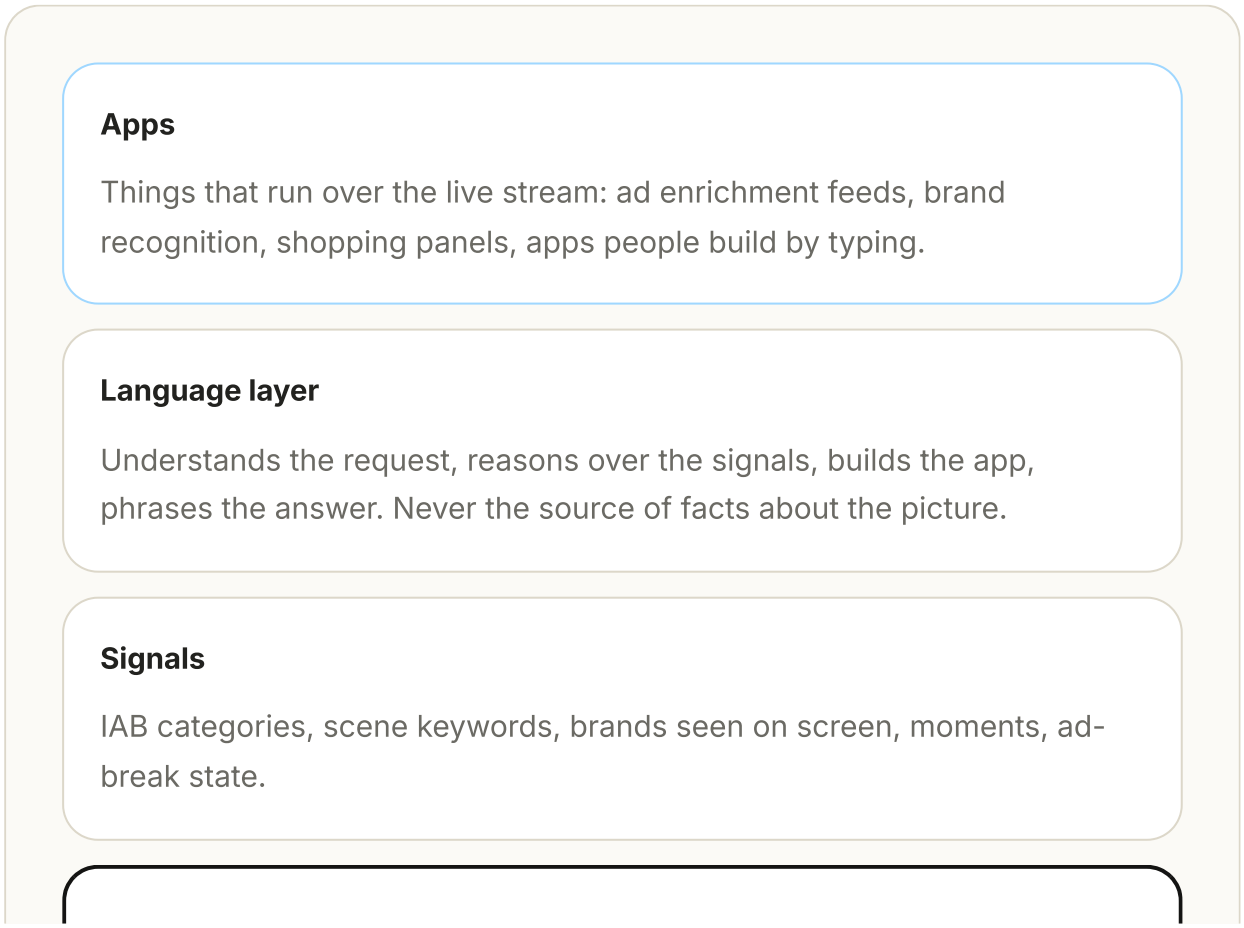
2. Related work

Modern vision language models connect visual encoders to language models in several ways. Flamingo interleaves visual features into a frozen language model through cross-attention [1]. BLIP-2 bridges a frozen image encoder and a frozen language model with a lightweight query transformer [2]. LLaVA connects a vision encoder to a language model and tunes the result on visual instructions [3, 4]. All build on the transformer [5] and on contrastive image–text pretraining such as CLIP [6]. Video extensions such as Video-LLaMA [7] and Video-ChatGPT [8] add temporal input.

A second line of work uses language models to compose visual tools. VisProg [9] and ViperGPT [10] have a language model produce a program that calls vision modules, and ReAct [11] and Toolformer [12] interleave reasoning with tool calls. Instruction following itself is shaped by work such as InstructGPT [13].

Our design borrows from both lines but fixes the direction of authority: the language model may plan, phrase and build, but statements about the picture come from the perception layer.

3. The layers



Vision language model (VLM): the top layer of understanding

Watches the stream, follows every moment, knows the ad break. Reads what's on screen and what's said.

Inputs

Live streams read once per channel at the source; 72-byte fingerprints from TVs on HDMI, free-to-air and apps.

We call the VLM the top layer because it sits at the top of the chain of authority, even though data flows upward from the inputs. Signals are its output. The language layer consumes signals, never raw guesses about pixels, and apps consume both.



Figure 1. The same idea as drawn in Edge's explainer film: the VLM on top, the language model underneath helping it reason.

4. Four rules that keep the layers apart

Rule 1. Perception stays with the VLM

Only the perception layer says what is on screen. Its answers are constrained to allowlisted vocabularies and strict answer shapes, and are validated before use (see Technical report 02). If the VLM returns nothing for a moment, the system has nothing to say about that moment.

Rule 2. The language layer handles intent and output

The language model's jobs are to understand what a person is asking for, to reason over signals, to build the app that should run, and to phrase answers. It can decide that an app should show a shopping prompt when a kitchen appliance is on screen. It cannot decide that a kitchen appliance is on screen.

Rule 3. Customer context enters before perception

Channel owners know things the model does not: which topics they want emphasised or avoided, which brands they work with, what language the audience speaks. In our design that context is turned into instructions that shape what the perception layer looks for and how it chooses, before it answers, rather than being used afterwards to edit the answer. This keeps the facts and the preferences in the right order: the model looks with the owner's guidance, and then reports what it saw.

Rule 4. The view is not the engine

The Control Room is a live view of what the engine is doing. It shows signals and app state as they happen. It is useful for operators and for demonstrations, but nothing in it is the source of a signal.

5. Cost: read once, recognise often

A layered design makes cost a matter of where each layer runs and how often.

- **Once per channel.** The VLM reads each live channel at the source. Every viewer of that channel shares the same reading.
- **Only when new.** Within a channel, a learned local vocabulary recognises familiar scenes without a model call. The VLM is called when the scene is new, and its answer teaches the vocabulary.
- **Lookups at the TV.** On a television, recognition is a 72-byte fingerprint and a lookup against clips the VLM has already fingerprinted. No model runs at view time (see Technical report 01).
- **Language when asked.** The language layer runs when someone asks for something or an app needs building, not on every frame.

The result is that model cost follows the number of channels and the rate of genuinely new scenes, not the number of screens.

6. From signals to apps

Because signals have a fixed, documented shape, apps can be built against them without touching the perception layer. An ad enrichment feed reads categories and keywords at the break. A brand recognition panel reads brands seen on screen. A shopping panel reads the scene and the products in it. In each case the app is a consumer of the VLM's answer, and the language layer helps build it from a typed description. We describe that process in Technical report 05.

7. Why this order matters

Putting perception first trades some flexibility for trust. A language-first system can say more about a scene, because it can fill gaps with plausible text. A perception-first system says less, but what it says is tied to the picture and to a fixed vocabulary that downstream systems understand. For advertising, where a category or a brand becomes part of a commercial decision, we think that trade is the right one.

8. Limitations and open questions

- **The boundary is a discipline, not a theorem.** Rules 1 and 2 are enforced by schemas, validators and code review. A new app that passes language-model text off as a perception result would break the design; we guard against that in review.
- **Context can mislead.** Customer context that enters before perception can bias what the model looks for. We treat programme hints as untrusted evidence and keep the model's grounding rules above them.
- **Not one set of weights.** "The VLM" in this report names the perception layer as a whole: sampling, speech transcription, local recognition, brand reading and model calls. It is not a claim that a single network does all of it.
- **Measurement.** We have not yet published a controlled comparison between perception-first and language-first designs on the same streams.

References

1. J.-B. Alayrac et al. Flamingo: a Visual Language Model for Few-Shot Learning. NeurIPS 2022. arXiv:2204.14198
2. J. Li, D. Li, S. Savarese and S. Hoi. BLIP-2. ICML 2023. arXiv:2301.12597
3. H. Liu, C. Li, Q. Wu and Y. J. Lee. Visual Instruction Tuning. NeurIPS 2023. arXiv:2304.08485

4. H. Liu, C. Li, Y. Li and Y. J. Lee. Improved Baselines with Visual Instruction Tuning. CVPR 2024. arXiv:2310.03744
5. A. Vaswani et al. Attention Is All You Need. NeurIPS 2017. arXiv:1706.03762
6. A. Radford et al. Learning Transferable Visual Models From Natural Language Supervision. ICML 2021. arXiv:2103.00020
7. H. Zhang, X. Li and L. Bing. Video-LLaMA. 2023. arXiv:2306.02858
8. M. Maaz, H. Rasheed, S. Khan and F. S. Khan. Video-ChatGPT. 2023. arXiv:2306.05424
9. T. Gupta and A. Kembhavi. Visual Programming: Compositional Visual Reasoning Without Training. CVPR 2023. arXiv:2211.11559
10. D. Surís, S. Menon and C. Vondrick. ViperGPT: Visual Inference via Python Execution for Reasoning. ICCV 2023. arXiv:2303.08128
11. S. Yao et al. ReAct: Synergizing Reasoning and Acting in Language Models. ICLR 2023. arXiv:2210.03629
12. T. Schick et al. Toolformer: Language Models Can Teach Themselves to Use Tools. NeurIPS 2023. arXiv:2302.04761
13. L. Ouyang et al. Training Language Models to Follow Instructions with Human Feedback. NeurIPS 2022. arXiv:2203.02155