Multimodal Understanding and Generation: DeepSeek-VL and Janus

Deep Dive into DeepSeek — Chapter 17

Yin Yang

Foundation Books Project

The one rule of this chapter

Understanding reads retained pixels and emits a checkable claim.
Generation reads a prompt and emits new pixels.

- The same scene, the same nouns, can support both tasks — but the input object, model branch, output type, and scorer all change between them.
- One eight-field record, $R_{\mathrm{mm}} = (A_{\mathrm{in}}, \Pi, M, b, x, d, y, S)$, carries both without hiding which fields differ.

The one rule of this chapter

Understanding reads retained pixels and emits a checkable claim.
Generation reads a prompt and emits new pixels.

- The same scene, the same nouns, can support both tasks — but the input object, model branch, output type, and scorer all change between them.
- One eight-field record, $R_{\mathrm{mm}} = (A_{\mathrm{in}}, \Pi, M, b, x, d, y, S)$, carries both without hiding which fields differ.

Goal: follow one shelf scene through DeepSeek-VL, DeepSeek-VL2, Janus, Janus-Pro, and JanusFlow — processor contracts, decoupled visual roles, two generation mechanisms, and output-matched evaluation — and see where each piece lives in the DeepSeek source.

Roadmap

1. One record, two output contracts

Roadmap

1. One record, two output contracts
2. Visual understanding: DeepSeek-VL and VL2

Roadmap

1. One record, two output contracts
2. Visual understanding: DeepSeek-VL and VL2
3. Image generation: Janus, Janus-Pro, JanusFlow

Roadmap

1. One record, two output contracts
2. Visual understanding: DeepSeek-VL and VL2
3. Image generation: Janus, Janus-Pro, JanusFlow
4. Evaluation, runtime, and failure discipline

One Record, Two Output Contracts

One record, eight fields

$$R_{\mathrm{mm}} = (A_{\mathrm{in}}, \Pi, M, b, x, d, y, S)$$

- R_U: $b =$ understand, y is text or localization tied to a nonempty A_{in}.
- R_G: $b \in \{\text{generate-AR}, \text{generate-flow}\}$, y is a raster.

One record, eight fields

$$R_{\mathrm{mm}} = (A_{\mathrm{in}}, \Pi, M, b, x, d, y, S)$$

- R_U: $b =$ understand, y is text or localization tied to a nonempty A_{in}.
- R_G: $b \in \{\text{generate-AR}, \text{generate-flow}\}$, y is a raster.

R_U and R_G are not competing tuples — they are the same eight-field record, specialized by which fields are populated and what type y carries.

One shelf, two specializations

Field	Shelf understanding R_U	Shelf generation R_G
A_{in}	Current shelf image, dimensions, hash	$\varnothing$ — text-to-image task
x	Count the blue notebooks right of the mug	Create the same scene on a white shelf
y, S	Text ($\perp$ before execution); exact-count contract	Raster ($\perp$ before execution); object/relation/label rubric

One shelf, two specializations

Field	Shelf understanding R_U	Shelf generation R_G
A_{in}	Current shelf image, dimensions, hash	$\varnothing$ — text-to-image task
x	Count the blue notebooks right of the mug	Create the same scene on a white shelf
y, S	Text ($\perp$ before execution); exact-count contract	Raster ($\perp$ before execution); object/relation/label rubric

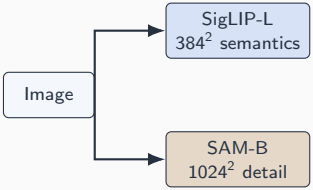
Regenerating the image creates a new A_{in} even when x is unchanged — understanding emits a claim about retained pixels; generation emits new pixels.

Visual Understanding: DeepSeek-VL and VL2

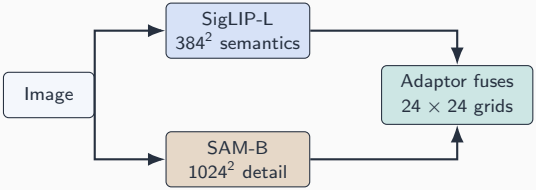
DeepSeek-VL: two fixed views, one fused budget



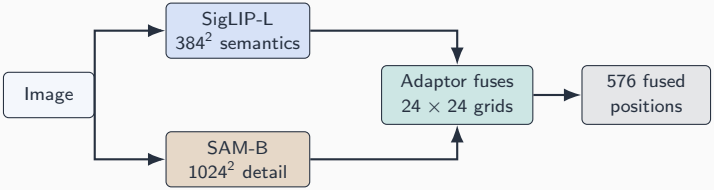
DeepSeek-VL: two fixed views, one fused budget



DeepSeek-VL: two fixed views, one fused budget

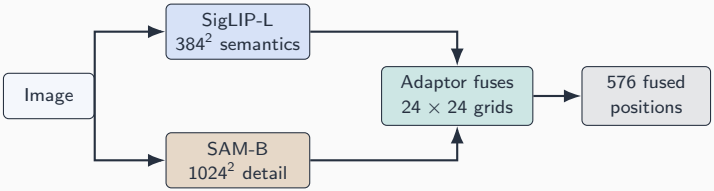


DeepSeek-VL: two fixed views, one fused budget



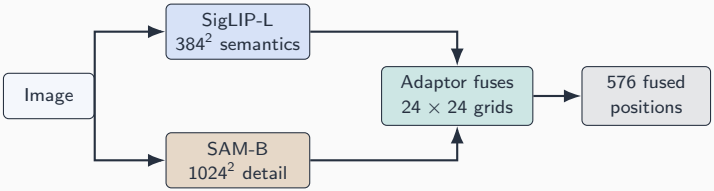
Output is $24 \cdot 24 = 576$ *fused* positions — not two concatenated sequences.

DeepSeek-VL: two fixed views, one fused budget



Output is $24 \cdot 24 = 576$ *fused* positions — not two concatenated sequences. VL2 replaces this fixed pair with a processor-selected local tile grid plus one global view, run through a single shared encoder, so its sequence length varies with the selected layout.

DeepSeek-VL: two fixed views, one fused budget

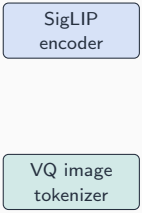


Output is $24 \cdot 24 = 576$ *fused* positions — not two concatenated sequences. VL2 replaces this fixed pair with a processor-selected local tile grid plus one global view, run through a single shared encoder, so its sequence length varies with the selected layout.

An exact visual-position claim must name the model family as well as the input resolution — "576" is a VL fact, not a universal one.

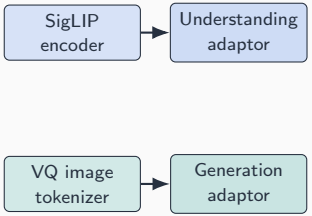
Image Generation: Janus, Janus-Pro, JanusFlow

Decoupled visual roles, one shared transformer



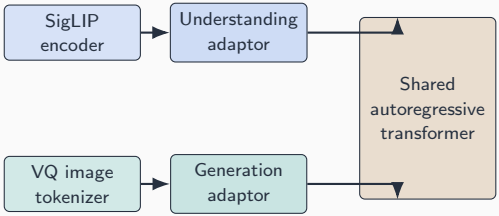
Janus-Pro keeps this architecture and changes training, data, and scale.

Decoupled visual roles, one shared transformer



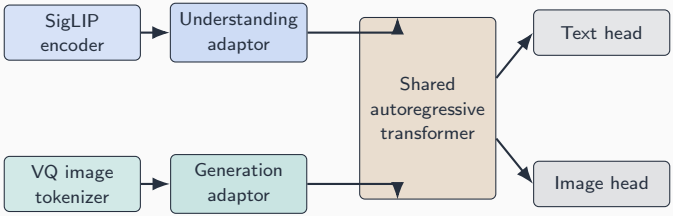
Janus-Pro keeps this architecture and changes training, data, and scale.

Decoupled visual roles, one shared transformer



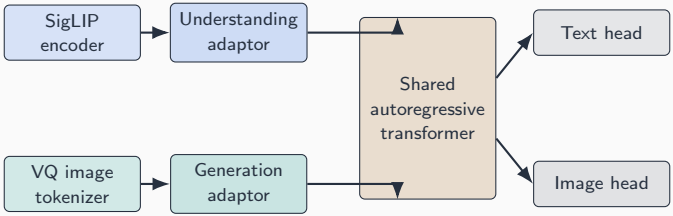
Janus-Pro keeps this architecture and changes training, data, and scale.

Decoupled visual roles, one shared transformer



Janus decouples visual *encoding* by task, while sharing the expensive sequence model.
Janus-Pro keeps this architecture and changes training, data, and scale.

Decoupled visual roles, one shared transformer



Janus decouples visual *encoding* by task, while sharing the expensive sequence model. Janus-Pro keeps this architecture and changes training, data, and scale.

Sharing the transformer never made the encoders, token spaces, losses, or scorers interchangeable between the two branches.

From concept to code: two visual roles, one class

from Janus/janus/models/modeling_vlm.py

```
class MultiModalityCausalLM(MultiModalityPreTrainedModel):
    def __init__(self, config):
        self.vision_model = vision_cls(**vision_config.params)
        self.aligner = aligner_cls(aligner_config.params)

        self.gen_vision_model = gen_vision_cls()
        self.gen_aligner = gen_aligner_cls(gen_aligner_config.params)
        self.gen_head = gen_head_cls(gen_head_config.params)

        self.language_model = LlamaForCausalLM(language_config)
```

From concept to code: two visual roles, one class

from Janus/janus/models/modeling_vlm.py

```
class MultiModalityCausalLM(MultiModalityPreTrainedModel):
    def __init__(self, config):
        self.vision_model = vision_cls(**vision_config.params)
        self.aligner = aligner_cls(aligner_config.params)

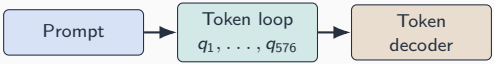
        self.gen_vision_model = gen_vision_cls()
        self.gen_aligner = gen_aligner_cls(gen_aligner_config.params)
        self.gen_head = gen_head_cls(gen_head_config.params)

        self.language_model = LlamaForCausalLM(language_config)
```

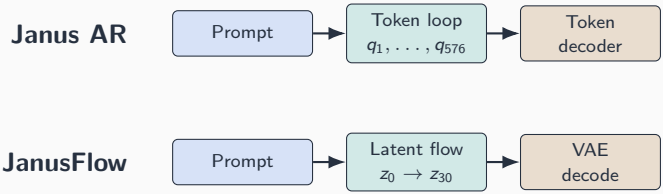
vision_model/aligner serve understanding; gen_vision_model/gen_aligner/ gen_head serve generation. One language_model is shared by both — the decoupling claim, as a constructor. [INSPECTED: Janus modeling_vlm.py]

Two generation mechanisms, one artifact type

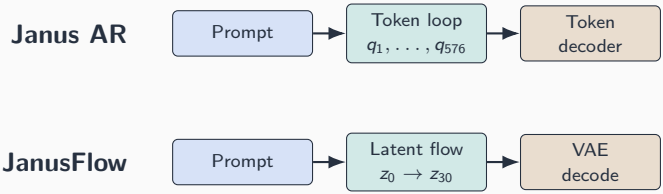
Janus AR



Two generation mechanisms, one artifact type

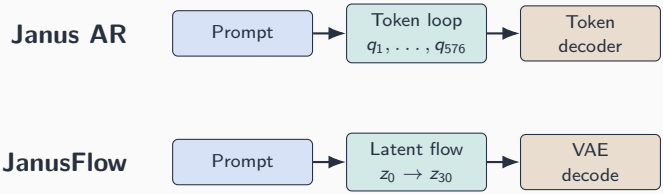


Two generation mechanisms, one artifact type



JanusFlow keeps the Janus understanding path and replaces discrete image-token sampling with a rectified-flow ODE: $v_{\text{cfg}} = v_{\text{uncond}} + w(v_{\text{cond}} - v_{\text{uncond}})$, $z_{n+1} = z_n + \Delta t\, v_{\text{cfg}}$.

Two generation mechanisms, one artifact type



JanusFlow keeps the Janus understanding path and replaces discrete image-token sampling with a rectified-flow ODE: $v_{\text{cfg}} = v_{\text{uncond}} + w(v_{\text{cond}} - v_{\text{uncond}})$, $z_{n+1} = z_n + \Delta t\, v_{\text{cfg}}$.

Step count and image-token count are not comparable denominators — one samples tokens, the other updates a continuous latent.

Evaluation, Runtime, and Failure Discipline

Output-matched scorers, not one pooled score

Benchmark	Janus baseline	Janus-Pro-7B
MMBench (understanding)	69.4	79.2
GenEval (generation)	0.61	0.80
DPG-Bench (generation)	79.68	84.19

[REPORTED: Janus-Pro technical report]

Output-matched scorers, not one pooled score

Benchmark	Janus baseline	Janus-Pro-7B
MMBench (understanding)	69.4	79.2
GenEval (generation)	0.61	0.80
DPG-Bench (generation)	79.68	84.19

These metrics share no probability scale, denominator, or checked output type. A ten-point MMBench gain and a 0.61 $\rightarrow$ 0.80 GenEval gain are not the same kind of improvement.

[REPORTED: Janus-Pro technical report]

Runtime work units are not comparable denominators

- Understanding: visual positions, text prefill, text decoding — $80 + 576 = 656$ toy positions.
- Autoregressive generation: image-code steps — 576 tokens per image in the repository example.
- Flow generation: batched solver steps — 30 steps, $B = 5$, equivalent to 150 sample-steps.

Runtime work units are not comparable denominators

- Understanding: visual positions, text prefill, text decoding — $80 + 576 = 656$ toy positions.
- Autoregressive generation: image-code steps — 576 tokens per image in the repository example.
- Flow generation: batched solver steps — 30 steps, $B = 5$, equivalent to 150 sample-steps.

These counts describe workload shape, not latency. The retained device campaigns are one-item compatibility and execution evidence — never a cross-device ranking.

Failure localization, stage by stage

Failure mechanism	Diagnostic
Perception	Compare the answer with the image at sufficient crop and resolution

Failure localization, stage by stage

Failure mechanism	Diagnostic
Perception	Compare the answer with the image at sufficient crop and resolution
Spatial grounding	Overlay boxes and test relations against object positions

Failure localization, stage by stage

Failure mechanism	Diagnostic
Perception	Compare the answer with the image at sufficient crop and resolution
Spatial grounding	Overlay boxes and test relations against object positions
Preprocessing	Inspect the processor transform and transformed image

Failure localization, stage by stage

Failure mechanism	Diagnostic
Perception	Compare the answer with the image at sufficient crop and resolution
Spatial grounding	Overlay boxes and test relations against object positions
Preprocessing	Inspect the processor transform and transformed image
Image generation	Inspect object count, geometry, and prompt relations

Failure localization, stage by stage

Failure mechanism	Diagnostic
Perception	Compare the answer with the image at sufficient crop and resolution
Spatial grounding	Overlay boxes and test relations against object positions
Preprocessing	Inspect the processor transform and transformed image
Image generation	Inspect object count, geometry, and prompt relations

Localize a failure to perception, processing, binding, generation, or scoring before choosing a mitigation — "the model was wrong" is not yet a diagnosis.

Concept-to-code map for the chapter

- **understanding entry point** → `Janus/inference.py`

Concept-to-code map for the chapter

- **understanding entry point** → `Janus/inference.py`
- **generation entry point** → `Janus/generation_inference.py`

Concept-to-code map for the chapter

- **understanding entry point** → `Janus/inference.py`
- **generation entry point** → `Janus/generation_inference.py`
- **DeepSeek-VL hybrid encoder** → `DeepSeek-VL/deepseek_vl/models/clip_encoder.py`

Concept-to-code map for the chapter

- **understanding entry point** → `Janus/inference.py`
- **generation entry point** → `Janus/generation_inference.py`
- **DeepSeek-VL hybrid encoder** → `DeepSeek-VL/deepseek_vl/models/clip_encoder.py`
- **VL2 tile and crop accounting** → `DeepSeek-VL2/deepseek_vl2/models/processing_deepseek_vl_v2.py`

Concept-to-code map for the chapter

- **understanding entry point** → `Janus/inference.py`
- **generation entry point** → `Janus/generation_inference.py`
- **DeepSeek-VL hybrid encoder** → `DeepSeek-VL/deepseek_vl/models/clip_encoder.py`
- **VL2 tile and crop accounting** → `DeepSeek-VL2/deepseek_vl2/models/processing_deepseek_vl_v2.py`
- **Janus decoupled encoder classes** → `Janus/janus/models/modeling_vlm.py`

Concept-to-code map for the chapter

- **understanding entry point** → `Janus/inference.py`
- **generation entry point** → `Janus/generation_inference.py`
- **DeepSeek-VL hybrid encoder** → `DeepSeek-VL/deepseek_vl/models/clip_encoder.py`
- **VL2 tile and crop accounting** → `DeepSeek-VL2/deepseek_vl2/models/processing_deepseek_vl_v2.py`
- **Janus decoupled encoder classes** → `Janus/janus/models/modeling_vlm.py`
- **JanusFlow guided Euler step** → `Janus/demo/app_janusflow.py`

Concept-to-code map for the chapter

- **understanding entry point** → `Janus/inference.py`
- **generation entry point** → `Janus/generation_inference.py`
- **DeepSeek-VL hybrid encoder** → `DeepSeek-VL/deepseek_vl/models/clip_encoder.py`
- **VL2 tile and crop accounting** → `DeepSeek-VL2/deepseek_vl2/models/processing_deepseek_vl_v2.py`
- **Janus decoupled encoder classes** → `Janus/janus/models/modeling_vlm.py`
- **JanusFlow guided Euler step** → `Janus/demo/app_janusflow.py`

One record, two output contracts, output-matched evaluation.

Next: the DeepSeek-OCR chapter follows a document page into plain text, markdown, table structure, or grounded geometry — the same object-first discipline, applied to a new typed output.