



SGLang-Diffusion

High-performance serving for offline and real-time diffusion

Yihao Wang & Kevin Mi | RadixArk | SGLang Team

PyTorch Conference 2026

github.com/sgl-project/sglang → python/sglang/multimodal_gen

SGLang: Widely-adopted Open-source Inference Framework

SGL

LLMs & VLMs

SGL-DIFFUSION

image, video, action, 3D & world model

SGL-OMNI

ASR & TTS



- ✓ High Performance
- ✓ One Engine & CLI for All Modalities
- ✓ Optimized for Agentic Workloads
- ✓ Broad Hardware Support:
NVIDIA, AMD, Intel, TPU, Trainium, ...
- ✓ Open Source

Many models. One serving system.

Image

Generation + editing

FLUX.1 / 2 / 3 · Qwen-Image
Z-Image · Krea-2
Ideogram 4 · GLM-Image

Video

Offline + streaming · World models

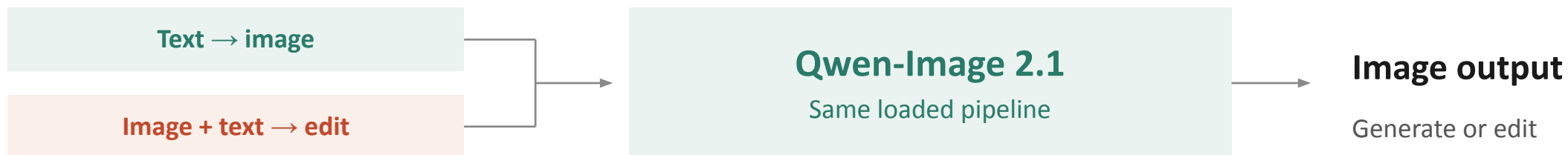
Wan · FastWan · causal-DMD Wan
LTX-2 / 2.3 / 2.5
MiniMax-H3 · FastH3 · VDN-H3
Cosmos3 · LingBot World
SANA-WM · LongLive2

Beyond pixels

3D · actions

Hunyuan3D
 π 0.5 · Cosmos3-Action
FLUX.3-Action

One loaded pipeline, generation + editing



Native task: T12I, with optional reference images.

Same engine, same CLI, different silicon

NVIDIA

H100 · H200 · A100 · B200 ·
RTX 4090 / 5090 / DGX Spark

AMD ROCm

MI300X · MI325X · MI355X
Triton attention + AITER kernels

Intel

XPU support

Ascend NPU

A2 · A3

Apple Silicon

M-series via MPS
Triton-free fallbacks
optional MLX fused norms

Moore Threads (MUSA)

MTT S5000
FA3 / Sage / SDPA fallback chain



```
uv pip install 'sglang[diffusion]' \
  --prerelease=allow
```

```
sglang serve \
  --model-path MiniMaxAI/MiniMax-H3 \
  --num-gpus 4 \
  --tp-size 2 \
  --ulysses-degree 2 \
  --performance-mode speed \
  --model-variant fl2va
```

```
sglang generate \
  --model-path Lightricks/LTX-2.5-Diffusers \
  --pipeline-class-name LTX2TwoStagePipeline \
  --prompt "A cinematic shot of a red fox." \
  --height 1088 --width 1920 \
  --save-output
```

One platform abstraction — kernel and fallback support is model- and platform-specific.

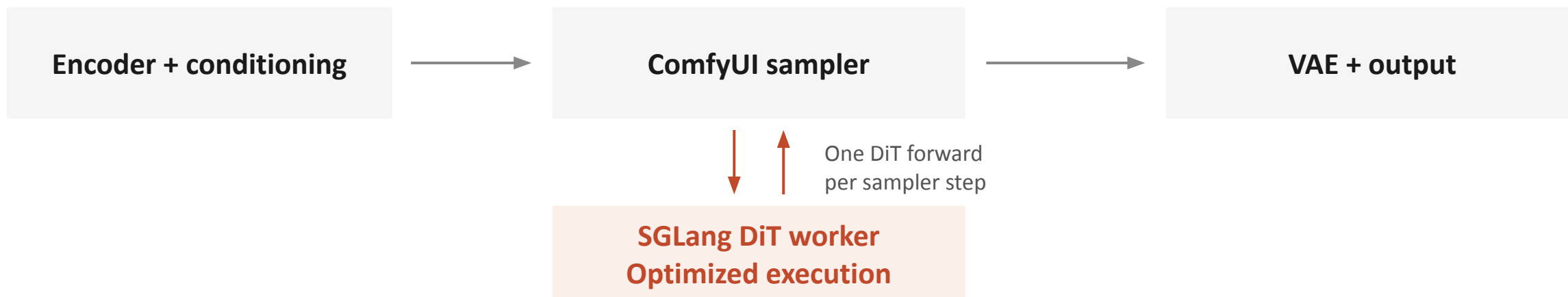
OpenAI-compatible API · CLI · Python SDK · ComfyUI

install guide: docs.sglang.io/docs/sglang-diffusion/installation

Flexible ComfyUI workflows with SGLang

Integrated mode keeps ComfyUI in control of encoding, sampling and decoding

COMFYUI WORKFLOW



Flexible workflows

Compose conditioning, sampler and VAE nodes.
Swap the model loader to use SGLang.

SGLang performance controls

Tune the DiT with SGLDOptions:
Parallelism, attention, quantization
Caching, execution and memory

Keep the workflow flexible. Accelerate the DiT with SGLang.

Integrated: FLUX, Z-Image, Qwen-Image / 2.1, MiniMax-H3. Compatibility depends on the model and workflow.

Diffusion benchmarks need an explicit workload

GB300 288GB | Single-request latency (s), lower is better | Data updated 2026-10-12

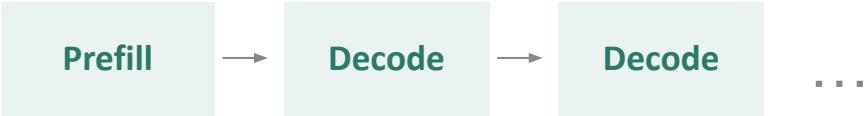
MODEL / TASK	WORKLOAD	SGLANG	FASTEST PEER	SPEEDUP
MiniMax-H3 T2V + audio	4x GB300 1344x768 124f 50 steps	29.714 s	32.536 s LightX2V	1.09x
FastH3 8-Step V2 T2V + audio distilled	4x GB300 1344x768 124f 8 steps trained VSA	5.478 s	7.170 s FastVideo	1.31x
Cosmos3-Nano Image-to-video	4x GB300 1280x720 189f 35 steps CFG 6	22.628 s	26.391 s vLLM-Omni	1.17x
Qwen-Image 2.1 Text-to-image	1x GB300 1024x1024 40 steps CFG 1	2.005 s	2.147 s vLLM-Omni	1.07x

Compare within each row: same workload, GPU count and model recipe.

Warm client-side medians; best lossless serving profile per model. No response cache / Cache-DiT / quantized checkpoints.
FastH3 uses its distilled checkpoint + trained VSA. Source: [diffusion-bench-framework](#)

Offline diffusion and LLM serving

LLM serving



Append one token per decode step

Offline diffusion



Refine the latent, then decode the result

State	Growing KV cache	Fixed-size latent, repeated updates
Hot path	Often bandwidth-bound	Often compute-heavy
Batching	Join between decode steps	Group compatible requests first
Quality	Sampling / deployment choices	Steps / precision / approximations

Different stages. Different memory and execution decisions.

Typical offline workloads; causal world models have a different state contract.

One pipeline, an ordered list of stages

A pipeline is a list of stages



Each stage declares the components it uses.

Different component lifetimes make residency and offload decisions possible.

Optimization stack

MEMORY PLACEMENT & PLANNING

- Component residency controls
- Layerwise offload + resident layers
- Warmup calibration and peak estimation
- Joint GPU / host memory planning
- Planner validation and rollback (in development)

EXECUTION PATH

- Stage disaggregation and worker pools
- Compatible request batching
- Exact conditioning cache reuse
- Parallelism, CUDA Graphs and replay correctness
- Kernels and communication / compute overlap

Placement first. Execution next. Quality-controlled approximations last.

Component residency and layerwise offload

Resident baseline

Keep frequently reused weights on GPU when memory permits.

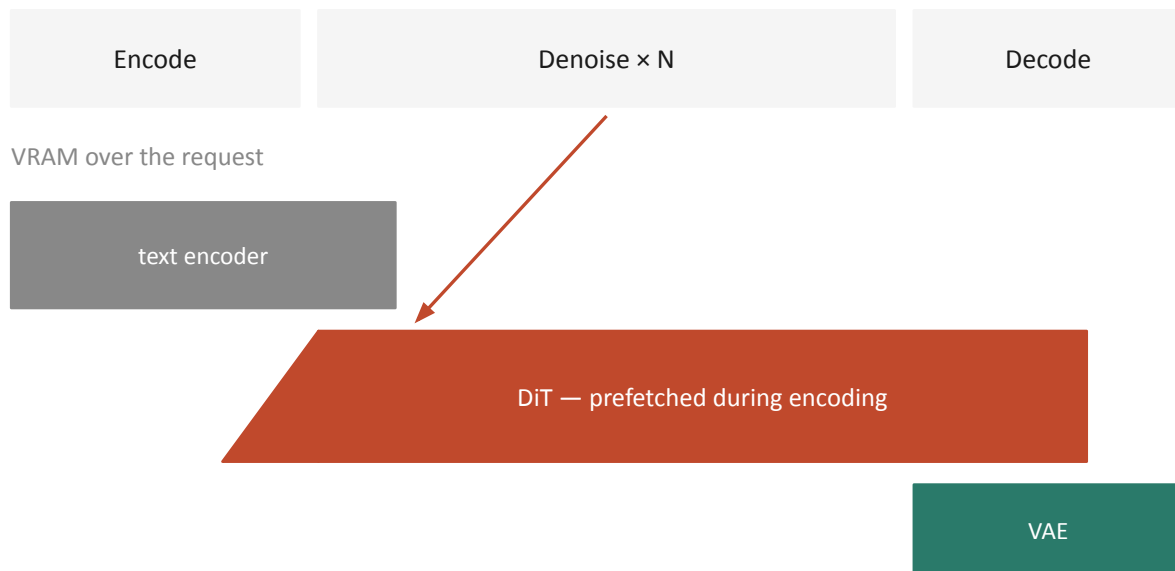
Avoid repeated transfers.

Component residency

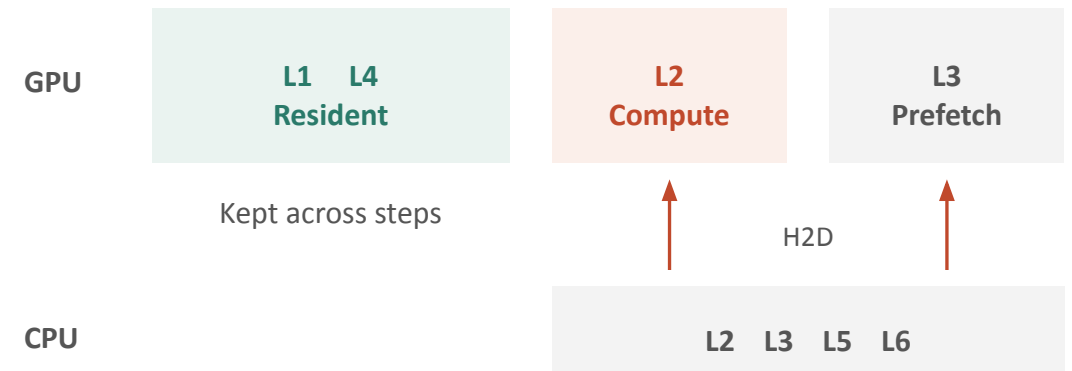
Move whole components between stages; each must fit while active.

Layerwise offload

Keep selected layers on GPU; stream and prefetch the rest.



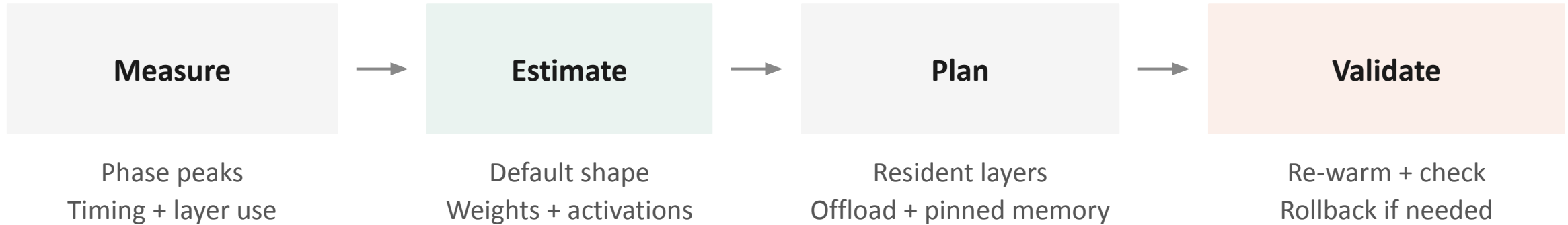
Layerwise offload + resident layers



More resident layers: fewer repeated transfers.

More prefetch: larger staging buffers.

A calibrated planner chooses memory placement



One plan across GPU and host memory

Phase peaks + safety reserve constrain VRAM.

Pinned-host and transition budgets constrain offload.

Objective: lower estimated latency within the available headroom.

Default workload, batch 1. Explicit placement settings remain fixed.

Warmup measurement: main. Solver + validation: open PRs #37917 / #37918 (2026-10-14).

Batch compatible requests before execution



Admission before execution

Compatible requests enter together; not mid-flight token-style batching.

ROADMAP · Continuous batching for diffusion models

Batch size competes with the same activation and workspace budget.

Dynamic batching guide. Check model and request compatibility; measure queue delay as well as throughput.

Cross-request reuse: conditioning outputs and prefix KV

EXACT CONDITIONING REUSE



Native cache: text embeddings, negative prompts, reference features

PREFIX KV REUSE VIA SRT



GLM-Image: external SRT backend, T2I only. Full prompts need not match.

Reuse matching work across requests; keep the generation recipe.

Parallelism and PyTorch execution

PARALLELISM

Encode

Fold: shard encoder weights across replica GPUs
 Batch DP: split prompts across encoder copies
 Use otherwise-idle DiT GPUs during encoding.

Denoise

SP: latent tokens (Ulysses / Ring / CP)
 TP: layer weights | CFG: cond / uncond branches

Decode

VAE decode parallelism, where supported

Auto selects per encoder; support is model-dependent.
 Validate numerics for each topology and batch shape.

EXECUTION PATH

Breakable CUDA Graph (Initiated by SGLang)

Launch-bound — ~30 blocks × ~30 steps × dozens of kernels;
 when the GPU is fast enough, the CPU issuing work becomes the bottleneck



breakable CUDA graph — capture supported DiT segments;
 varlen packing, backend dispatch and SP collectives stay eager

Warmup primes setup. BCG reduces recurring launch overhead.

Model-aware attention backends

sparse-video families · Sage · FA3 / FA4 · SDPA · ...

Pipeline Ulysses all-to-all with attention

Split attention heads into groups; overlap peer copies with computation.

SEQUENTIAL

One exposed chain

QKV all-to-all

Attention: all heads

Output all-to-all

PIPELINED

IPC peer copies

Copy engine

Q0

Q1

Q2

O0

O1

O2

SMs

Attn g0

Attn g1

Attn g2

Q: QKV exchange O: output exchange g: head group

time

Different resources

Peer copies use the copy engine.
Attention uses the SMs.

Byte-wise check per shape;
otherwise stay sequential.

FLUX 2048² | -24.2% time

1.90 → 1.44 ms

LTX-2 shape | -16.6% time

3.85 → 3.21 ms

Hide transfer time,
not model work.

Quality as a serving parameter

Quantization

Low-precision paths

FP8 · ModelOpt FP8 / NVFP4 · MXFP4 / 8 · bitsandbytes
· modelslim
+ a KV-cache path for causal video

Temporal cache

TeaCache · Cache-DiT · Spectrum

reuse or forecast block outputs instead of computing the step

Progressive resolution

early steps at low resolution, then DCT-space upsample
— the expensive steps run on a smaller latent

Tier guarantees are relative to deployment settings. Independent precision and cache flags still apply.

Request quality contract

--quality = bit-exact — selected deployment's reference path

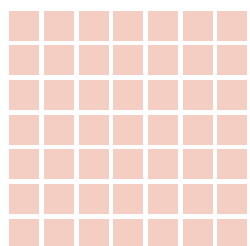
--quality = lossless (default) — eligible request-gated DiT / VAE kernel fusions

--quality = high — adds model-owned approximate paths



VDN-H3 scales on an optimized SGLang pipeline

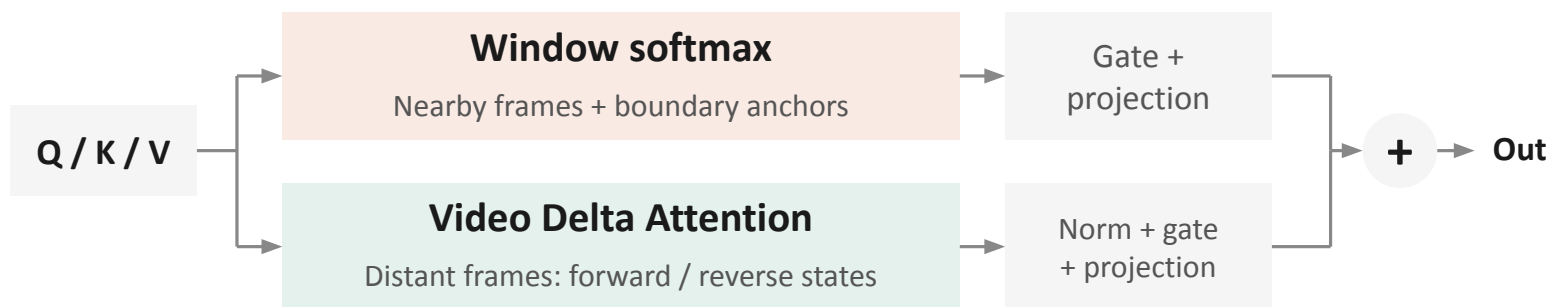
H3: dense attention



All video frames
attend to each other

Video-to-video attention

VDN: local detail + distant context



Text/audio interactions retain softmax. Video-path schematic; not a layer-by-layer map.

VDN-H3: hybrid attention + 8-forward distillation

14.4s video in 8.7s

8 denoising steps; ~8.7s full request after warmup

6.9s denoising • 768p/24 fps • 8× B200

Up to 1.60× faster per forward

86–91% parallel efficiency vs. 56–64% reference

Same B200 host, 2–8 GPUs; reference FP8 / SGLang MXFP8

Key optimizations:

- **Chunked delta scans with fewer launches**
Less recurrent state traffic and host launch work
- **MXFP8 GEMMs + fused activation quantization**
Lower-precision GEMMs with fused input conversion
- **Overlap attention branches and Ulysses all-to-all**
Overlap communication with independent branch work

New attention backends reuse the same scaling and decode optimizations

[Source: H3 cookbook · 1344×768, 345 frames, 24 fps · model usage rights apply](#)

VDN-H3: faster-than-playback video generation

SG L VDN-H3 on SGLang-Diffusion continuous stream · 8x B200 generation 9.0s < playback 14.4s STREAM 00:00



The main video player shows a street scene with a silver car (license plate WGE K 655) and a white Mercedes SUV (license plate WGE K 409) driving. A yellow duck is in the foreground. The video is labeled 'Shot 01' and 'NOW PLAYING'. The status bar at the bottom shows 'NOW PLAYING Shot 01' and a progress indicator '0:00.0 / 0:14.4'.

GENERATION QUEUE

- Shot 01: NOW PLAYING, 14.4s
- Shot 02: GENERATING, 0%
- Shot 03: QUEUED, waiting

FASTER THAN PLAYBACK

9.0 s < 14.4 s

generation < video duration
Shown demo: 8x B200

Live streaming

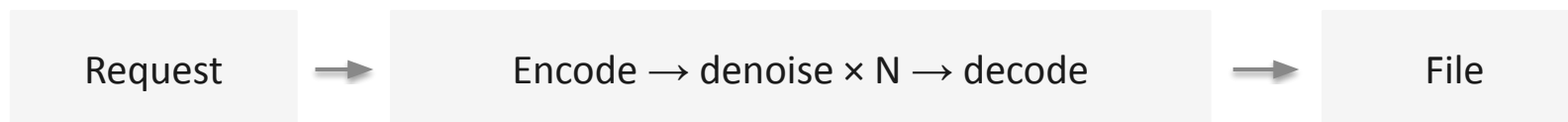
Generate the next segment
while the current one plays.

Interactive videos

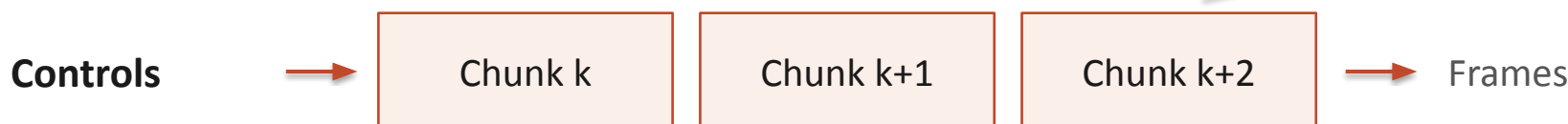
Respond to new input
between video segments.

Offline optimizes throughput. Realtime optimizes the next frame.

OFFLINE



REALTIME / AUTOREGRESSIVE



Session state

Causal KV
Conditioning
Decoder history

Causal KV-cache quantization

LingBot World: QVG INT4 / INT2, opt-in

Older completed chunks compressed; current and recent chunks stay BF16.

Less resident KV memory, with quality and packing-latency trade-offs.

CLOSE

See it, then run it

Many thanks to the great community!



GALLERY
More samples & benchmark results



SGLang Diffusion
Documentation & Cookbook

Yihao Wang & Kevin Mi