Package 'diffuseR'

August 8, 2026

Title Functional Interface to Diffusion Models in R

Version 0.2.2

Description A native R implementation of diffusion models providing a functional interface to state-of-the-art generative AI. Inspired by the 'Python' library 'diffusers' from 'Hugging Face' <https://huggingface.co/>, 'diffuseR' generates and manipulates images from text prompts using models such as 'Stable Diffusion', with no 'Python' dependency. Supports multiple diffusion schedulers and device acceleration.

License Apache License (>= 2)

URL https://github.com/cornball-ai/diffuseR

BugReports https://github.com/cornball-ai/diffuseR/issues

Encoding UTF-8

Imports torch, jsonlite, grid, png, jpeg

Suggests av, hfhub, safetensors, simplermarkdown, tinytest

VignetteBuilder simplermarkdown

RoxygenNote 7.3.3

NeedsCompilation no

Author Troy Hernandez [aut, cre] (ORCID: <https://orcid.org/0009-0005-4248-604X>), cornball.ai [cph], The HuggingFace Team [cph] (portions ported from the diffusers library (Apache-2.0); see inst/COPYRIGHTS), Lightricks Ltd. [cph] (LTX checkpoint layout and pipeline constants; see inst/COPYRIGHTS)

Maintainer Troy Hernandez <troy@cornball.ai>

Repository CRAN

Date/Publication 2026-08-08 12:40:02 UTC

Contents

audio_encode_ltx23 *Audio Conditioning Frontend for LTX-2.3*

Description

Turns user audio into the normalized, packed audio latents the joint denoiser conditions on (lip sync): decode to 16 kHz stereo PCM, log-mel via a causal STFT (filter 1024, hop 160, 64 slaney-normed mel bins to 8 kHz — the checkpoint's preprocessing spec), then the audio VAE encoder in argmax mode. The STFT and mel-filterbank constructors were verified against the checkpoint's stored vocoder bases (identical up to bf16 rounding), so the convention matches training.

audio_vae_ltx23 *LTX-2.3 Audio VAE*

Description

Fresh R port of the LTX-2 audio autoencoder from the diffusers reference (Apache-2.0, autoencoder_kl_ltx2_audio.py), configured per the checkpoint: pixel norm, height-axis causality, base 128 channels with multipliers (1, 2, 4), 8 latent channels, 64 mel bins, no attention. The decoder produces mel for the vocoder; the encoder turns user audio into conditioning latents (lip sync).

auto_devices *Auto-Configure Device Assignment*

Description

Automatically determines optimal device configuration for diffusion model components based on available VRAM (via nvidia-smi) and GPU architecture.

Usage

```
auto_devices(model = "sdxl", strategy = "auto")
```

Arguments

model	Character. Model type: "sd21" or "sdxl".
strategy	Character. Memory strategy: "auto" (default), "full_gpu", "unet_gpu", or "cpu_only". See Details.

Details

Strategies:

"auto" Detect free VRAM and choose the best strategy

"full_gpu" All components on CUDA (10GB+ free VRAM for SDXL)

"unet_gpu" Only unet on CUDA, rest on CPU (6GB+ for SDXL)

"cpu_only" All components on CPU

On Blackwell GPUs (RTX 50xx), "unet_gpu" is forced due to TorchScript compatibility issues, regardless of available VRAM. The native modules (‘use_native_unet‘ and friends) do not have this restriction.

Value

A named list of device assignments suitable for ‘models2devices()‘.

Examples

```
# Force a strategy: no GPU or nvidia-smi needed.
str(auto_devices("sdxl", strategy = "cpu_only"))

str(auto_devices("sd21", strategy = "unet_gpu"))

# Auto-detect free VRAM and pick a strategy for this machine.
str(auto_devices("sdxl"))
```

`bpe_tokenizer` *BPE Tokenizer*

Description

Native R implementation of Byte-Pair Encoding tokenizer. Loads from HuggingFace tokenizer.json format.

Usage

```
bpe_tokenizer(tokenizer_path)
```

Arguments

`tokenizer_path`	Path to tokenizer.json or directory containing it.

Value

A bpe_tokenizer object.

checkpoint_flux *FLUX Checkpoint Readers*

Description

FLUX transformers ship in the diffusers layout: a directory with `config.json`, one or more `diffusion_pytorch_model*.sa` shards, and (when sharded) a `diffusion_pytorch_model.safetensors.index.json` weight map. These helpers open that layout behind the same checkpoint interface as `ltx23_open_checkpoint`, so the LTX group loaders and quantization machinery work unchanged. FLUX module names mirror the checkpoint keys 1:1 - no key mapping is needed.

checkpoint_ltx23 *LTX-2.3 Single-File Checkpoint Reader*

Description

LTX 2.3 checkpoints ship as one safetensors file containing every component (transformer, connectors, video VAE, audio VAE, vocoder), with the model version and full component configuration embedded in the safetensors metadata. These helpers open the file, validate the version, split the key space by component, and stream tensors into R torch modules one at a time so the 46 GB file is never fully materialized in memory.

clear_vram *Clear VRAM Cache*

Description

Forces garbage collection and clears CUDA memory cache.

Usage

```
clear_vram(verbose = FALSE)
```

Arguments

verbose	Logical. Print memory status before/after.

Value

Invisibly returns NULL.

Examples

```
if (torch::torch_is_installed()) {
  clear_vram()
}
```

CLIPTokenizer *Tokenize a prompt*

Description

Tokenize a prompt

Usage

```
CLIPTokenizer(prompt,
              merges = system.file("tokenizer/merges.txt", package = "diffuseR"),
           vocab_file = system.file("tokenizer/vocab.json", package = "diffuseR"),
               pad_token = 0L)
```

Arguments

prompt	A character string prompt describing the image to generate.
merges	Path to the merges file (BPE merges).
vocab_file	Path to the vocabulary file (token->id mapping).
pad_token	The token ID used for padding (default is 0).

Value

A 2D torch tensor of shape c(1, 77) containing the token IDs.

clip_pooled_output *Pooled CLIP output at the EOS position*

Description

The HF CLIPTextModel pooler_output: the final-layer-norm hidden state at the EOS token position, located by argmax over the token ids (EOS is the highest id in the CLIP vocab, and causal attention makes any padding after it irrelevant). No text projection is applied - this is what FLUX uses as pooled_projections.

Usage

```
clip_pooled_output(hidden_states, input_ids)
```

Arguments

hidden_states	Final-LN hidden states [B, S, D] from text_encoder_native (with apply_final_ln = TRUE).
input_ids	Token ids [B, S] (0-based, as fed to the encoder).

Value

Tensor [B, D].

`condition_ltx23` *LTX-2.3 Prefix Conditioning (image-to-video, video continuation)*

Description

Fresh R port of the frame-conditioning mechanics from the diffusers reference (Apache-2.0, pipelines/ltx2/pipeline_ltx2_imag and pipeline_ltx2_condition.py), restricted to prefix conditioning at latent index 0 with strength 1: a single start image (i2v) or the leading pixel frames of a previous clip (continuation). Conditioned latent tokens are initialized from the VAE-encoded pixels, see a per-token timestep of zero, and are frozen through the Euler loop.

`connectors_ltx23` *LTX-2.3 Text Embedding Connectors*

Description

Fresh R port of the LTX text connectors from the diffusers reference (Apache-2.0, src/diffusers/pipelines/ltx2/connectors.py). The connectors take raw stacked per-layer Gemma3 hidden states [batch, seq, caption_channels, num_layers + 1], normalize and project them per modality, replace padding with learnable registers, and run a small 1D transformer per modality to produce the DiT text embeddings.

`convert_sd21_pt_to_diffusers`
Convert cornball SD 2.1 TorchScript weights to a diffusers artifact

---

Description

Rebuilds a diffusers-layout directory (`unet/`, `vae/`, `text_encoder/`) from the cornball-ai/sd21-R TorchScript component `.pt` files, so the native safetensors pipeline (`download_sd21` / `sd_pipeline_from_safetensors`) can load SD 2.1 with no TorchScript.

Usage

```
convert_sd21_pt_to_diffusers(pt_dir = NULL, output_dir = NULL,
                             dtype = c("float16", "float32"), verbose = TRUE)
```

Arguments

pt_dir	Directory holding unet-cpu.pt, decoder-cpu.pt, text_encoder-cpu.pt (default: the diffuseR sd21 data location).
output_dir	Output diffusers directory (default: the sd_pipeline_from_safetensors / download_sd21 location).
dtype	"float16" (default, the hosted tier) or "float32".
verbose	Logical.

Details

A TorchScript trace preserves the exact parameter tensors, so the result is bit-identical to the source at the chosen dtype. This is the provenance-clean way to build the hosted artifact: the upstream stabilityai/stable-diffusion-2-1 repo was deprecated, SD 2.1 is CreativeML OpenRAIL++-M (redistributable), and cornball already hosts these weights as .pt. At float16 the components are all sub-2 GB single files (unet ~1.7 GB, text_encoder ~0.65 GB, vae ~0.16 GB), so they load on stock CRAN safetensors.

Value

Invisibly, output_dir.

---

ddim_scheduler_create *Create a DDIM Scheduler*

Description

Creates a Denoising Diffusion Implicit Models (DDIM) scheduler for use with diffusion models. DDIM schedulers provide a deterministic sampling process that offers faster inference compared to DDPM while maintaining high quality outputs.

Usage

```
ddim_scheduler_create(num_train_timesteps = 1000, num_inference_steps = 50,
                      eta = 0,
                      beta_schedule = c("linear", "scaled_linear", "cosine"),
                      beta_start = 0.00085, beta_end = 0.012,
                      rescale_betas_zero_snr = FALSE,
                      dtype = torch::torch_float32(),
                      device = torch::torch_device("cpu"))
```

Arguments

num_train_timesteps	Integer. The number of diffusion steps used to train the model. Default: 1000
num_inference_steps	Integer. The number of diffusion steps used for inference. Fewer steps typically means faster inference at the cost of sample quality. Default: 50

`eta` Numeric. Controls the amount of stochasticity. When eta=0, the sampling process is deterministic. When eta=1, the sampling process is equivalent to DDPM. Default: 0

`beta_schedule` Character. The beta schedule to use. Options are:

"linear" Linear beta schedule from beta_start to beta_end

"scaled_linear" Scaled linear schedule, generally gives better results

"cosine" Cosine schedule that approaches zero smoothly

Default: "linear"

`beta_start` Numeric. The starting value for the beta schedule. Default: 0.00085

`beta_end` Numeric. The final value for the beta schedule. Default: 0.012

`rescale_betas_zero_snr`
Logical. If TRUE, rescales the beta values

`dtype` The data type to use for computations. Default is torch_float32(). Options are torch_float16() and torch_float32().

`device` The device to use for computations. Options are torch_device("cpu"), torch_device("cuda").

Details

DDIM (Denoising Diffusion Implicit Models) was introduced by Song et al. (2020) as an extension to DDPM (Denoising Diffusion Probabilistic Models). It offers a deterministic sampling process and allows for controlling the number of inference steps independently from the training process.

The scheduler contains the noise schedule and methods for computing alpha, beta, and other parameters used in the diffusion process.

Value

A DDIM scheduler object that can be used with diffusion models to generate samples.

References

Song, J., Meng, C., & Ermon, S. (2020). "Denoising Diffusion Implicit Models." https://arxiv.org/abs/2010.02502

Examples

```
if (torch::torch_is_installed()) {
  scheduler <- ddim_scheduler_create(
    num_train_timesteps = 1000,
    num_inference_steps = 5,
    eta = 0.5,
    beta_schedule = "scaled_linear"
  )
  scheduler$timesteps
}
```

ddim_scheduler_step *Perform a DDIM scheduler step*

Description

Performs a single denoising step using the DDIM (Denoising Diffusion Implicit Models) algorithm. This function takes the output from a diffusion model at a specific timestep and computes the previous (less noisy) sample in the diffusion process.

Usage

```
ddim_scheduler_step(model_output, timestep, sample, schedule, eta = 0,
                    use_clipped_model_output = FALSE, thresholding = FALSE,
                    generator = NULL, variance_noise = NULL,
                    clip_sample = FALSE, set_alpha_to_one = FALSE,
                    prediction_type = c("epsilon", "sample", "v_prediction"),
                    dtype = torch::torch_float32(), device = "cpu")
```

Arguments

model_output	Numeric array. The output from the diffusion model, typically representing predicted noise or the denoised sample depending on 'prediction_type'.
timestep	Integer. The current timestep in the diffusion process.
sample	Numeric array. The current noisy sample at timestep 't'.
schedule	List. The DDIM scheduler object containing the necessary parameters created from ddim_scheduler_create()
eta	Numeric. Controls the stochasticity of the process. When eta=0, DDIM is deterministic. When eta=1, it's equivalent to DDPM. Default: 0
use_clipped_model_output	Logical. Whether to clip the model output before computing the sample update. Can improve stability. Default: FALSE
thresholding	Logical. Whether to apply thresholding to the output. Default: FALSE
generator	An optional random number generator for reproducibility. Default: NULL
variance_noise	Optional pre-generated noise for the variance when eta > 0. If NULL and eta > 0, noise will be generated. Default: NULL
clip_sample	Logical. Whether to clip the sample. Default: FALSE
set_alpha_to_one	Logical. Whether to override the final alpha value to 1. Used for numerical stability in the final step. Default: FALSE
prediction_type	Character. The type of prediction the model outputs. Options are: **"epsilon"** The model predicts the noise **"sample"** The model predicts the denoised sample directly

	"v_prediction" The model predicts the velocity vector (v)
	Default: "epsilon"
`dtype`	The data type to use for computations. Default is torch_float32().
`device`	The device to use for computations. Options are "cpu" and "cuda".

Details

The DDIM step function implements the core sampling algorithm of DDIM described in Song et al. 2020. It computes the previous sample x_t-1 given the current sample x_t and the model output.

The algorithm differs from DDPM by using a non-Markovian diffusion process that allows for deterministic sampling and fewer inference steps without sacrificing quality.

When using ‘prediction_type="epsilon"‘ (most common), the model predicts the noise that was added to create the current noisy sample. For ‘prediction_type="sample"‘, the model predicts the clean sample directly. The ‘v_prediction‘ option implements the v-parameterization from Salimans & Ho (2022).

Value

A list containing:

‘prev_sample‘ The less noisy sample at timestep t-1

‘pred_original_sample‘ The predicted denoised sample

References

Song, J., Meng, C., & Ermon, S. (2020). "Denoising Diffusion Implicit Models." https://arxiv.org/abs/2010.02502

Salimans, T., & Ho, J. (2022). "Progressive Distillation for Fast Sampling of Diffusion Models." https://arxiv.org/abs/2202.00512

Examples

```
if (torch::torch_is_installed()) {
  scheduler <- ddim_scheduler_create(num_inference_steps = 5)
  sample <- torch::torch_randn(c(1, 4, 8, 8))
  model_output <- torch::torch_randn(c(1, 4, 8, 8))
  result <- ddim_scheduler_step(
    model_output = model_output,
    timestep = scheduler$timesteps[1],
    sample = sample,
    schedule = scheduler,
    eta = 0, # Deterministic sampling
    prediction_type = "epsilon")
  result$shape
}
```

decode_bpe *Decode token IDs to text*

Description

Decode token IDs to text

Usage

```
decode_bpe(tokenizer, ids, skip_special_tokens = TRUE)
```

Arguments

tokenizer	A bpe_tokenizer object.
ids	Integer vector or matrix of token IDs.
skip_special_tokens	Logical. Skip special tokens in output.

Value

Character string or vector.

dit_flux *FLUX Transformer (MMDiT)*

Description

Fresh R port of FluxTransformer2DModel from the diffusers reference implementation (Apache-2.0, src/diffusers/models/transformers/transformer_flux.py). The module tree mirrors the diffusers state-dict keys 1:1, so checkpoints load without remapping. FLUX.1-schnell has no guidance embedder (guidance_embeds = FALSE); the guidance-distilled dev variant is not implemented.

dit_flux2 *FLUX.2 Transformer (MMDiT)*

Description

Fresh R port of Flux2Transformer2DModel from the diffusers reference implementation (Apache-2.0, src/diffusers/models/transformers/transformer_flux2.py). Defaults are the klein-4B configuration (5 double + 20 single blocks). Guidance embeddings (FLUX.2-dev) are not implemented; klein is step-distilled with `guidance_embeds = false`. Timestep conditioning has no pooled-text component, and the three modulation projections are shared across all blocks.

`dit_flux2_modules` *FLUX.2 Transformer Building Blocks*

Description

Fresh R port of the FLUX.2 MMDiT blocks from the diffusers reference implementation (Apache-2.0, src/diffusers/models/transformers/transformer_flux2.py). Key differences from FLUX.1: modulation is computed ONCE at model level by shared `flux2_modulation` projections and passed into the blocks (block norms are parameterless), feed-forwards use SwiGLU with the gate fused into `linear_in`, the single-stream block is a ViT-22B-style parallel block with fully fused projections, and every linear is bias-free. Module field names mirror the diffusers state-dict keys 1:1. Reuses `flux_attention` (bias = FALSE), `ltx23_rms_norm`, `.ltx23_sdpa`, and `flux_apply_rotary_emb`.

`dit_flux_modules` *FLUX Transformer Building Blocks*

Description

Fresh R port of the FLUX MMDiT blocks from the diffusers reference implementation (Apache-2.0, src/diffusers/models/transformers/transformer_flux.py and src/diffusers/models/normalization.py). Module field names mirror the diffusers state-dict keys 1:1 so checkpoints load without remapping. Reuses the LTX primitives `ltx23_rms_norm`, `.ltx23_sdpa` and `ltx23_feed_forward`.

`dit_ltx23` *LTX-2.3 Audio-Video Diffusion Transformer*

Description

Fresh R port of the LTX-2 transformer from the diffusers reference (Apache-2.0, src/diffusers/models/transformers/transforme
configured for LTX 2.3: gated attention, cross-attention modulation, prompt AdaLN, split RoPE, and connector-projected text embeddings (no in-model caption projection).

`dit_ltx23_modules` *LTX-2.3 Transformer Building Blocks*

Description

Fresh R port of the LTX-2 transformer components from the diffusers reference (Apache-2.0, src/diffusers/models/transformers/ transformer_ltx2.py and shared modules). Field names mirror the diffusers module tree so checkpoint keys map 1:1.

dit_zimage_modules *Z-Image Transformer Block Modules*

Description

Fresh R port of the Z-Image DiT building blocks from the diffusers reference (Apache-2.0, src/diffusers/models/transformers/
Z-Image is a single-stream DiT: text and image tokens share one sequence and one set of block weights. Each block uses sandwich RMSNorms (a learned norm before AND after both the attention and the feed-forward) and a scale/gate-only modulation — four chunks (scale_msa, gate_msa, scale_mlp, gate_mlp), no shift, gates tanh-squashed, scales 1 + x. The attention is plain joint self-attention, so the FLUX attention module is reused with bias = FALSE and eps = 1e-5.

download_component *Download a single TorchScript model component*

Description

Downloads a specific model component file (e.g., UNet, decoder, text encoder) using `hfhub::hub_download()` from the cornball-ai dataset repos.

Usage

```
download_component(model_name = "sd21", component, device = "cpu",
                   overwrite = FALSE, show_progress = TRUE)
```

Arguments

`model_name`	Character string, the name of the model (e.g., `"sd21"`).
`component`	Character string, the component to download (e.g., `"unet"`, `"decoder"`).
`device`	Character string, the device type (e.g., `"cpu"` or `"cuda"`).
`overwrite`	Logical; if `TRUE`, force re-download even if cached.
`show_progress`	Logical; if `TRUE` (default), displays progress during download.

Value

The local file path to the downloaded component (character string).

Examples

```
## Not run:
path <- download_component("sd21", "text_encoder", "cpu")

## End(Not run)
```

download_flux — *Download and Prepare FLUX.1-schnell Weights*

Description

Downloads FLUX.1-schnell from HuggingFace (weights Apache-2.0, but the repo is gated behind a license click-through) and quantizes the 12B transformer to a local NF4 (~7 GB) or fp8 (~12 GB) artifact.

download_flux1 — *Download FLUX.1-schnell and build the quantized artifact*

Description

Skips work already done: a valid quantized manifest short-circuits the transformer download; cached files are not re-fetched. Needs `HF_TOKEN` set for the gated repo (see the error message it raises without one). The bf16 transformer source (~24 GB in the HuggingFace cache) may be deleted after quantization.

Usage

```
download_flux1(quantize = TRUE, precision = c("nf4", "fp8"), output_dir = NULL,
               text_encoders = TRUE, verbose = TRUE)
```

Arguments

`quantize`	Logical. Build the quantized artifact after downloading.
`precision`	"nf4" (~7 GB, GPU-resident on 16 GB cards) or "fp8" (~12 GB, CPU-resident, streamed; near-bf16 quality).
`output_dir`	Directory for the quantized artifact.
`text_encoders`	Logical. Also fetch the CLIP + T5 text encoders, tokenizer, VAE, and scheduler config (~10 GB).
`verbose`	Logical.

Value

Invisibly, a list with `transformer_dir`, `artifact_dir`, and `support` (named file paths).

download_flux2 — *Download and Prepare FLUX.2 Klein 4B Weights*

Description

Downloads FLUX.2-klein-4B from HuggingFace (Apache-2.0, ungated) and quantizes the 4B transformer to a local fp8 (~4 GB) or NF4 (~2.3 GB) artifact.

download_flux2_klein *Download FLUX.2-klein-4B and build the quantized artifact*

Description

Skips work already done: a valid quantized manifest short-circuits the transformer download; cached files are not re-fetched. No token is needed (the repo is ungated). The bf16 transformer source (~7.8 GB in the HuggingFace cache) may be deleted after quantization.

Usage

```
download_flux2_klein(quantize = TRUE, precision = c("auto", "fp8", "nf4"),
                     output_dir = NULL, text_encoders = TRUE, verbose = TRUE)
```

Arguments

quantize	Logical. Build the quantized artifact.
precision	"auto" (default: fp8 when safetensors supports float8, else nf4), "fp8" (~4 GB, GPU-resident; near-bf16 quality), or "nf4" (~2.3 GB).
output_dir	Directory for the quantized artifact.
text_encoders	Logical. Also fetch the Qwen3 text encoder, tokenizer, VAE, and scheduler config (~8.3 GB).
verbose	Logical.

Value

Invisibly, a list with `transformer_dir`, `artifact_dir`, and `support` (named file paths).

download_ltx2 *Download the LTX-2.3 checkpoint and build a quantized artifact*

Description

Skips work that is already done: a valid manifest short-circuits everything; a cached 46 GB source skips the download. The source file may be deleted after quantization (it is never removed automatically).

Usage

```
download_ltx2(quantize = TRUE, precision = c("nf4", "fp8"), output_dir = NULL,
              text_encoder = TRUE, verbose = TRUE)
```

Arguments

quantize	Logical. Build the quantized artifact after downloading.
precision	"nf4" (~19 GB, readable by every safetensors) or "fp8" (~26 GB, needs float8 write support).
output_dir	Directory for the artifact. NULL derives it from precision.
text_encoder	Logical. Also fetch the Gemma3 text encoder and tokenizer (~25 GB, shared with LTX-2.0; from the Lightricks/LTX-2 repo).
verbose	Logical.

Details

Both quantized tiers are buildable here. recommend returns nf4 for LTX on any card with 14 GB or more (it prefers nf4 at 1280 px over fp8 at 1024 px, since video trades weight precision for resolution), so nf4 is the tier most users want. fp8 additionally needs a safetensors that can *write* float8; asking for it without one warns and builds nf4 instead rather than failing inside the quantizer.

Value

Invisibly, a list with checkpoint (source path or NULL), artifact_dir, precision, text_encoder_dir, and fp8_dir for back-compatibility – the artifact directory when precision is "fp8", NULL otherwise, since a field named fp8_dir pointing at an nf4 artifact would be a trap.

---

download_ltx23 *Download and Prepare LTX-2.3 Model Weights*

Description

Downloads the LTX-2.3 distilled checkpoint (46 GB, LTX-2 Community License) and the Gemma3 text encoder from HuggingFace with an explicit consent prompt, then quantizes the transformer to the local fp8 artifact (~26 GB) used by the GPU-poor pipeline.

---

download_model *Download TorchScript model files for Stable Diffusion*

Description

Downloads the required model files (e.g., UNet, decoder, text encoder) for a given Stable Diffusion model using hfhub::hub_download().

Usage

```
download_model(model_name = "sd21",
               devices = list(unet = "cpu", decoder = "cpu", text_encoder = "cpu"),
                 unet_dtype_str = NULL, overwrite = FALSE, show_progress = TRUE,
                 download_models = FALSE)
```

Arguments

model_name	Name of the model (e.g., "sd21" for stable-diffusion-2-1)
devices	Either a single device string or a named list with elements 'unet', 'decoder', 'text_encoder'; optionally 'encoder'
unet_dtype_str	Optional: "float16" or "float32" (only applies if unet uses CUDA)
overwrite	If TRUE, force re-download of model files
show_progress	Show download progress messages
download_models	If TRUE, download the model files from HuggingFace

Details

Files are cached by hfhub (typically `~/.cache/huggingface/hub/`). Legacy files in the old `R_user_dir()` location are also recognized.

Value

A named list of full file paths, keyed by component name.

Examples

```
## Not run:
paths <- download_model("sd21")

## End(Not run)
```

---

download_sd21 *Download the Stable Diffusion 2.1 diffusers weights*

Description

Fetches the UNet, VAE, and CLIP text encoder from the `cornball-ai/sd21-R` HuggingFace dataset (fp16 diffusers safetensors, converted from the original OpenRAIL weights; the upstream `stabilityai` repo was deprecated). About 2.5 GB, one-time. The native tokenizer and DDIM scheduler need no downloads.

Usage

```
download_sd21(verbose = TRUE)
```

Arguments

verbose	Logical.

Value

Invisibly, the diffusers directory (the parent of `unet/`, `vae/`, `text_encoder/`).

download_sdxl *Download the Stable Diffusion XL diffusers weights*

Description

Fetches the UNet (re-sharded to sub-2 GB shards), VAE, and both CLIP text encoders from the `cornball-ai/sdxl-R` HuggingFace dataset (fp16 diffusers safetensors, converted from the original `stabilityai/stable-diffusion-xl-base-1.0` OpenRAIL++ weights). About 7 GB, one-time. The native tokenizer and DDIM scheduler need no downloads.

Usage

```
download_sdxl(verbose = TRUE)
```

Arguments

`verbose`	Logical.

Value

Invisibly, the diffusers directory (the parent of `unet/`, `vae/`, `text_encoder/`, `text_encoder_2/`).

download_zimage *Download and Prepare Z-Image-Turbo Weights*

Description

Downloads Z-Image-Turbo from HuggingFace (Apache-2.0, ungated) and quantizes the 6B transformer to a local fp8 (~6.3 GB) or NF4 (~3.6 GB) artifact. The checkpoint ships the transformer in float32 (24.6 GB), so the one-time quantize saves a lot of disk and load time.

download_zimage_turbo *Download Z-Image-Turbo and build the quantized artifact*

Description

Skips work already done: a valid quantized manifest short-circuits the transformer download; cached files are not re-fetched. No token is needed (the repo is ungated). The float32 transformer source (~24.6 GB in the HuggingFace cache) may be deleted after quantization.

Usage

```
download_zimage_turbo(quantize = TRUE, precision = c("auto", "fp8", "nf4"),
                      output_dir = NULL, text_encoders = TRUE, verbose = TRUE)
```

Arguments

quantize	Logical. Build the quantized artifact.
precision	"auto" (default: fp8 when safetensors supports float8, else nf4), "fp8" (~6.3 GB, GPU-resident; near-bf16 quality), or "nf4" (~3.6 GB).
output_dir	Directory for the quantized artifact.
text_encoders	Logical. Also fetch the Qwen3-4B text encoder, tokenizer, VAE, and scheduler config (~8.2 GB).
verbose	Logical.

Value

Invisibly, a list with `transformer_dir`, `artifact_dir`, and `support` (named file paths).

---

encode_bpe *Encode text to token IDs*

Description

Encode text to token IDs

Usage

```
encode_bpe(tokenizer, text, add_special_tokens = TRUE, max_length = NULL,
           padding = "none", truncation = FALSE, return_tensors = "list")
```

Arguments

tokenizer	A bpe_tokenizer object.
text	Character string or vector to encode.
add_special_tokens	Logical. Add BOS/EOS tokens.
max_length	Integer. Maximum sequence length (NULL for no limit).
padding	Character. Padding strategy: "none", "max_length", or "longest".
truncation	Logical. Truncate to max_length.
return_tensors	Character. Return type: "list" or "pt" (torch tensors).

Value

List with input_ids and attention_mask.

encode_qwen *Encode prompts with the Qwen tokenizer*

Description

With `chat_template = TRUE` each prompt is wrapped as a single user turn with the generation prompt, matching `apply_chat_template(..., add_generation_prompt = TRUE)`. With `enable_thinking = FALSE` (the FLUX.2 klein pipeline behavior) the template closes with an empty thinking block; with `enable_thinking = TRUE` (the Z-Image pipeline behavior) it ends at the assistant turn. Right-pads with `<|endoftext|>`.

Usage

```
encode_qwen(tokenizer, texts, max_length = 512L, chat_template = TRUE,
            enable_thinking = FALSE)
```

Arguments

`tokenizer`	A `qwen_bpe_tokenizer`.
`texts`	Character vector of prompts.
`max_length`	Integer. Fixed sequence length (klein: 512). NULL for no truncation/padding.
`chat_template`	Logical. Wrap in the Qwen3 chat template.
`enable_thinking`	Logical. Leave the model's thinking enabled (no empty think block). Default FALSE.

Value

List with `input_ids` and `attention_mask` integer matrices [length(texts), max_length] (ragged lists when `max_length` is NULL). Ids are 0-based.

encode_unigram *Encode text with a Unigram tokenizer*

Description

Normalizes (strip-right, multi-space collapse, control whitespace to space), applies the Metaspace pre-tokenizer, segments each pre-token by Viterbi over the Unigram scores, fuses consecutive unknowns, and appends EOS. T5 semantics: right padding with `<pad>` (id 0), truncation to `max_length - 1` before the EOS.

Usage

```
encode_unigram(tokenizer, texts, max_length = 256L, add_eos = TRUE, pad = TRUE)
```

Arguments

tokenizer	A unigram_tokenizer.
texts	Character vector of prompts.
max_length	Integer. Fixed sequence length (NULL for no truncation/padding).
add_eos	Logical. Append the EOS token.
pad	Logical. Right-pad to max_length.

Value

List with input_ids and attention_mask, each an integer matrix [length(texts), max_length] (or ragged lists when max_length is NULL). Ids are 0-based (HuggingFace convention); add 1 for R torch embedding lookups.

---

encode_with_gemma3 *Encode text with Gemma3 for LTX-2*

Description

Full pipeline for encoding text prompts using Gemma3 text encoder. Returns the raw stacked per-layer hidden states (embedding layer plus all transformer layers) for downstream connector modules, which handle normalization and projection themselves.

Usage

```
encode_with_gemma3(prompts, model = NULL, tokenizer = NULL,
                   max_sequence_length = 1024L, device = "cuda",
                   dtype = "float16", verbose = TRUE)
```

Arguments

prompts	Character vector of prompts.
model	Gemma3 text model (or path to load from).
tokenizer	Gemma3 tokenizer (or path to load from).
max_sequence_length	Integer. Maximum sequence length.
device	Character. Device for computation.
dtype	Character. Data type.
verbose	Logical. Print progress.

Value

List with prompt_embeds (raw stacked hidden states, shape [batch, seq_len, hidden_size, num_layers + 1]) and prompt_attention_mask.

encode_with_qwen3 *Encode prompts with the Qwen3 encoder for FLUX.2*

Description

Tokenizes with the chat template, runs the encoder with the padding mask, and concatenates the requested mid-stack hidden states per token, matching Flux2KleinPipeline._get_qwen3_prompt_embeds.

Usage

```
encode_with_qwen3(prompts, model, tokenizer, max_sequence_length = 512L,
                  out_layers = c(9L, 18L, 27L), device = NULL)
```

Arguments

prompts	Character vector.
model	A qwen3_encoder.
tokenizer	A qwen_bpe_tokenizer.
max_sequence_length	Integer. Fixed token length (klein: 512).
out_layers	Integer vector. Hidden-state layers (klein-4B: 9, 18, 27).
device	Device for the input ids (defaults to the model's).

Value

Tensor [length(prompts), max_sequence_length, 3 * hidden_size].

encode_with_t5 *Encode prompts with the T5 encoder*

Description

Tokenizes with encode_unigram (right padding to max_sequence_length) and runs the encoder. Matching the FLUX reference pipeline, no attention mask is used.

Usage

```
encode_with_t5(prompts, model, tokenizer, max_sequence_length = 256L,
               device = NULL)
```

Arguments

prompts	Character vector.
model	A t5_encoder.
tokenizer	A unigram_tokenizer.
max_sequence_length	Integer. Fixed token length (schnell: 256).
device	Device for the input ids (defaults to the model's).

Value

Tensor [length(prompts), max_sequence_length, d_model].

---

filename_from_prompt *Generate a filename from a prompt*

Description

This function generates a filename from a prompt by removing all non-alphanumeric characters and replacing them with underscores. The filename is limited to 50 characters. If 'datetime' is set to TRUE, the current date and time are prepended to the filename.

Usage

```
filename_from_prompt(prompt, datetime = TRUE)
```

Arguments

prompt	A character string representing the prompt.
datetime	Logical indicating whether to prepend the current date and time to the filename. Default is TRUE.

Value

A character string representing the generated filename.

Examples

```
filename_from_prompt("A beautiful sunset over the mountains")
filename_from_prompt("A beautiful sunset over the mountains", datetime = FALSE)
```

flowmatch_calculate_shift *Calculate shift for dynamic shifting*

Description

Computes the shift parameter (mu) based on sequence length for resolution-dependent timestep shifting.

Usage

```
flowmatch_calculate_shift(seq_len, base_seq_len = 256L, max_seq_len = 4096L,
                          base_shift = 0.5, max_shift = 1.15)
```

Arguments

seq_len	Integer. The sequence length (num_patches).
base_seq_len	Integer. Base sequence length. Default: 256
max_seq_len	Integer. Maximum sequence length. Default: 4096
base_shift	Numeric. Base shift value. Default: 0.5
max_shift	Numeric. Maximum shift value. Default: 1.15

Value

Numeric. The computed shift value (mu).

flowmatch_scale_noise *Scale noise for flow matching forward process*

Description

Applies the forward process in flow-matching: interpolates between the clean sample and noise.

Usage

```
flowmatch_scale_noise(sample, timestep, noise, schedule)
```

Arguments

sample	torch tensor. The clean sample.
timestep	torch tensor. The current timestep.
noise	torch tensor. The noise tensor.
schedule	List. The FlowMatch scheduler object.

Value

torch tensor. The noisy sample at timestep t.

`flowmatch_scheduler_create` *Create a FlowMatch Euler Discrete Scheduler*

Description

Creates a FlowMatch scheduler for use with flow-matching diffusion models like LTX-2. FlowMatch schedulers use Euler integration for sampling, which is simpler and often faster than DDIM-style schedulers.

Usage

```
flowmatch_scheduler_create(num_train_timesteps = 1000L, shift = 1,
                           use_dynamic_shifting = FALSE, base_shift = 0.5,
                           max_shift = 1.15, base_seq_len = 256L,
                           max_seq_len = 4096L, invert_sigmas = FALSE,
                           shift_terminal = NULL,
                           time_shift_type = c("exponential", "linear"))
```

Arguments

`num_train_timesteps`	Integer. The number of diffusion steps used to train the model. Default: 1000
`shift`	Numeric. The shift value for the timestep schedule. Default: 1.0
`use_dynamic_shifting`	Logical. Whether to apply timestep shifting on-the-fly based on the image/video resolution. Default: FALSE
`base_shift`	Numeric. Value to stabilize generation. Increasing reduces variation. Default: 0.5
`max_shift`	Numeric. Maximum shift allowed. Increasing encourages more variation. Default: 1.15
`base_seq_len`	Integer. Base sequence length for dynamic shifting. Default: 256
`max_seq_len`	Integer. Maximum sequence length for dynamic shifting. Default: 4096
`invert_sigmas`	Logical. Whether to invert the sigmas (used by some models like Mochi). Default: FALSE
`shift_terminal`	Numeric or NULL. End value of shifted schedule. Default: NULL
`time_shift_type`	Character. Type of dynamic shifting: "exponential" or "linear". Default: "exponential"

Details

FlowMatch (Flow Matching) is a framework for training continuous normalizing flows by regressing onto target probability paths. The Euler discrete scheduler implements simple Euler integration for sampling from trained flow models.

The core update rule is: `prev_sample = sample + dt * model_output` where `dt = sigma_next - sigma_current`.

Value

A FlowMatch scheduler object (list) containing:

sigmas The noise schedule

timesteps The timestep schedule

num_train_timesteps Training timesteps

config All configuration parameters

References

Lipman, Y., Chen, R. T. Q., Ben-Hamu, H., Nickel, M., & Le, M. (2022). "Flow Matching for Generative Modeling." https://arxiv.org/abs/2210.02747

Examples

```
if (torch::torch_is_installed()) {
  scheduler <- flowmatch_scheduler_create(
    num_train_timesteps = 1000,
    shift = 1.0
  )

  # Set timesteps for inference
  scheduler <- flowmatch_set_timesteps(scheduler, num_inference_steps = 8)
  scheduler$timesteps
}
```

`flowmatch_scheduler_step` *Perform a FlowMatch scheduler step*

Description

Performs a single denoising step using Euler integration. This is the core sampling function for FlowMatch models.

Usage

```
flowmatch_scheduler_step(model_output, timestep, sample, schedule,
                         generator = NULL)
```

Arguments

model_output	torch tensor. The output from the diffusion model (velocity prediction).
timestep	Numeric. The current timestep.
sample	torch tensor. The current noisy sample.
schedule	List. The FlowMatch scheduler object.
generator	torch generator or NULL. Random generator for reproducibility.

Details

The FlowMatch Euler step is remarkably simple: `prev_sample = sample + dt * model_output` where `dt = sigma_next - sigma_current`.

This implements the Euler method for solving the probability flow ODE in continuous normalizing flows.

Value

A list containing:

prev_sample The denoised sample at the previous timestep

schedule The updated scheduler with incremented step_index

---

flowmatch_set_timesteps

Set timesteps for inference

Description

Configures the scheduler timesteps for a specific number of inference steps. This must be called before using the scheduler for denoising.

Usage

```
flowmatch_set_timesteps(schedule, num_inference_steps = 50L, device = "cpu",
                        mu = NULL, sigmas = NULL, timesteps = NULL)
```

Arguments

schedule	List. The FlowMatch scheduler object.
num_inference_steps	Integer. Number of denoising steps. Default: 50
device	Character or torch device. Device for tensors. Default: "cpu"
mu	Numeric or NULL. Shift parameter for dynamic shifting. Required if use_dynamic_shifting is TRUE. Default: NULL
sigmas	Numeric vector or NULL. Custom sigma values. Default: NULL
timesteps	Numeric vector or NULL

Value

Updated scheduler with configured timesteps and sigmas.

`flux2_bn_normalize` *Normalize patchified latents with the VAE BatchNorm statistics*

Description

FLUX.2 has no scalar scaling/shift factor; latents are standardized per packed channel with the VAE's `bn.running_mean` / `bn.running_var` (eps 1e-4). Reference: encode/decode paths of Flux2KleinPipeline.

Usage

```
flux2_bn_normalize(latents, bn_mean, bn_var, eps = 1e-04, inverse = FALSE)
```

Arguments

`latents`	Tensor [B, 128, H, W] (patchified).
`eps`	Numeric. BatchNorm epsilon.
`inverse`	Logical. De-normalize (decode path) instead.
`bn_mean, bn_var`	Float tensors [128].

Value

Tensor like `latents`.

`flux2_double_block` *FLUX.2 double-stream (MMDiT) block*

Description

Image and text streams with externally supplied (shift, scale, gate) modulation triples, joint attention (txt first), and SwiGLU feed-forwards. Reference: Flux2TransformerBlock.

Usage

```
flux2_double_block(dim, num_attention_heads, attention_head_dim, mlp_ratio = 3,
                   eps = 1e-06, bias = FALSE)
```

Arguments

`dim`	Integer. Model dimension.
`num_attention_heads`	Integer. Attention heads.
`attention_head_dim`	Integer. Per-head dimension.
`mlp_ratio`	Numeric. FF multiplier (FLUX.2: 3.0).
`eps`	Numeric. Norm epsilon.
`bias`	Logical.

Value

Module whose forward(hidden_states, encoder_hidden_states, temb_mod_img, temb_mod_txt, image_rotary_emb) returns `list(encoder_hidden_states, hidden_states)`.

`flux2_empirical_mu` *Empirical timestep shift for FLUX.2*

Description

BFL's piecewise-linear fit of the dynamic-shifting mu as a function of image sequence length and step count; replaces FLUX.1's calculate_shift. Reference: compute_empirical_mu (adapted from BFL sampling.py).

Usage

```
flux2_empirical_mu(image_seq_len, num_steps)
```

Arguments

`image_seq_len`	Integer. Packed image token count.
`num_steps`	Integer. Inference steps.

Value

Numeric mu for `flowmatch_set_timesteps`.

flux2_feed_forward *FLUX.2 feed-forward (fused SwiGLU)*

Description

`linear_in` projects to twice the inner dim; SwiGLU gates the first half with SiLU and multiplies by the second half; `linear_out` projects back. Reference: Flux2FeedForward + Flux2SwiGLU.

Usage

```
flux2_feed_forward(dim, dim_out = NULL, mult = 3, bias = FALSE)
```

Arguments

`dim`	Integer. Input dimension.
`dim_out`	Integer. Output dimension (defaults to `dim`).
`mult`	Numeric. Inner dim multiplier (FLUX.2: 3.0).
`bias`	Logical.

Value

Module whose forward(x) returns the SwiGLU-gated projection of `x`, a tensor with the last axis of width `dim_out`.

flux2_is_quant_key *Test whether a FLUX.2 key is in the quantization cast set*

Description

Test whether a FLUX.2 key is in the quantization cast set

Usage

```
flux2_is_quant_key(key)
```

Arguments

`key`	Character vector of parameter names (diffusers-style).

Value

Logical vector.

`flux2_load_pipeline` *Load the FLUX.2 klein pipeline*

Description

Loads the quantized transformer artifact plus the FLUX.2 VAE decoder, Qwen3 text encoder, and tokenizer from the HuggingFace cache populated by `download_flux2_klein`. With fp8 precision the ~4 GB transformer stays GPU-resident.

Usage

```
flux2_load_pipeline(model_dir = NULL, device = "cuda",
                    precision = c("auto", "fp8", "nf4", "bf16"),
                    text_device = NULL, attn_chunk = NULL,
                    phase_offload = TRUE, pin = NULL, verbose = TRUE)
```

Arguments

`model_dir`	Quantized artifact directory (default: the `download_flux2_klein` location for `precision`), or a raw diffusers transformer directory.
`device`	Character. Compute device.
`precision`	"auto" (default: reuse an existing artifact, else fp8 when safetensors supports float8, else nf4), "fp8", or "nf4".
`text_device`	Device for the Qwen3 encoder (default: `device`; it encodes in its own phase and offloads).
`attn_chunk`	Integer or NULL. Attention query-chunk override.
`phase_offload`	Logical. One GPU tenant per phase.
`pin`	Logical or NULL. Page-lock the phase-swapped weights for DMA-rate transfer (see `staging`). NULL (default) resolves via `options(diffuseR.pin_staging)` then the host-RAM-aware `recommend` decision.
`verbose`	Logical.

Value

A `flux2_pipeline` list.

flux2_modulation *FLUX.2 shared modulation projection*

Description

`linear(silu(temb))` producing `mod_param_sets` triples of (shift, scale, gate). Computed once per forward at model level and broadcast to every block. Reference: Flux2Modulation.

Usage

```
flux2_modulation(dim, mod_param_sets = 2L, bias = FALSE)
```

Arguments

`dim`	Integer. Model dimension.
`mod_param_sets`	Integer. Number of (shift, scale, gate) triples.
`bias`	Logical.

Value

Module whose forward(temb) returns the modulation tensor `linear(silu(temb))`, holding `mod_param_sets` triples of (shift, scale, gate) along the last axis.

flux2_pack_latents *Pack patchified FLUX.2 latents into tokens*

Description

[B, C, H, W] -> [B, H * W, C] (row-major spatial flatten, channels-last). Reference: Flux2KleinPipeline._pack_latents.

Usage

```
flux2_pack_latents(latents)
```

Arguments

`latents`	Tensor [B, C, H, W].

Value

Tensor [B, H * W, C].

flux2_parallel_self_attention

FLUX.2 parallel self-attention (single-stream)

Description

ViT-22B-style parallel block internals: one fused projection produces QKV and the SwiGLU MLP input; one fused projection consumes cat(attention output, MLP output). Reference: Flux2ParallelSelfAttention + Flux2ParallelSelfAttnProcessor.

Usage

```
flux2_parallel_self_attention(query_dim, heads, dim_head, mlp_ratio = 3,
                              eps = 1e-06, bias = FALSE)
```

Arguments

query_dim	Integer. Model dimension.
heads	Integer. Attention heads.
dim_head	Integer. Per-head dimension.
mlp_ratio	Numeric. MLP hidden multiplier (FLUX.2: 3.0).
eps	Numeric. RMS norm epsilon.
bias	Logical.

Value

Module whose forward(hidden_states, image_rotary_emb, chunk_size) returns the block output [B, S, query_dim]: attention and MLP branches computed in parallel from one fused projection, concatenated, and projected back by a second fused layer.

flux2_patchify_latents

Patchify FLUX.2 latents (2x2 -> channels)

Description

[B, C, H, W] -> [B, 4C, H/2, W/2], channel order (C, ph, pw). Reference: Flux2KleinPipeline._patchify_latents.

Usage

```
flux2_patchify_latents(latents)
```

Arguments

latents	Tensor [B, C, H, W]; H and W must be even.

Value

Tensor [B, C * 4, H / 2, W / 2].

`flux2_prepare_latent_ids` *Build FLUX.2 latent position ids*

Description

Columns (T, H, W, L) with H and W carrying the packed-grid position (row-major: H varies slowest), T = L = 0. Reference: Flux2KleinPipeline._prepare_latent_ids.

Usage

```
flux2_prepare_latent_ids(height, width, device = "cpu")
```

Arguments

`height`	Integer. Packed grid height (pixel height / 16).
`width`	Integer. Packed grid width (pixel width / 16).
`device`	Device for the resulting tensor.

Value

Float tensor [height * width, 4].

`flux2_prepare_text_ids` *Build FLUX.2 text position ids*

Description

Columns (T, H, W, L) with only L varying: 0..len-1. Reference: Flux2KleinPipeline._prepare_text_ids.

Usage

```
flux2_prepare_text_ids(len, device = "cpu")
```

Arguments

`len`	Integer. Text sequence length.
`device`	Device for the resulting tensor.

Value

Float tensor [len, 4].

flux2_single_block *FLUX.2 single-stream block (parallel)*

Description

Parameterless LayerNorm with external modulation, then the fused parallel attention+MLP. Operates on the pre-concatenated [text; image] sequence (the reference model concatenates once before the stack). Reference: Flux2SingleTransformerBlock.

Usage

```
flux2_single_block(dim, num_attention_heads, attention_head_dim, mlp_ratio = 3,
                   eps = 1e-06, bias = FALSE)
```

Arguments

dim	Integer. Model dimension.
num_attention_heads	Integer. Attention heads.
attention_head_dim	Integer. Per-head dimension.
mlp_ratio	Numeric. MLP multiplier (FLUX.2: 3.0).
eps	Numeric. Norm epsilon.
bias	Logical.

Value

Module whose forward(hidden_states, temb_mod, image_rotary_emb) returns the joint hidden states.

flux2_transformer *FLUX.2 transformer model*

Description

Shared modulation computed once per forward; double blocks over separate text/image streams, then single (parallel) blocks over the concatenated [text; image] sequence. Rotary embeddings are precomputed by the caller with flux_pos_embed (axes_dim = c(32, 32, 32, 32), theta = 2000) over the concatenated [text; image] 4-axis position ids.

Usage

```
flux2_transformer(in_channels = 128L, num_layers = 5L, num_single_layers = 20L,
                  attention_head_dim = 128L, num_attention_heads = 24L,
                  joint_attention_dim = 7680L, mlp_ratio = 3,
                  timestep_guidance_channels = 256L,
                  axes_dims_rope = c(32L, 32L, 32L, 32L), rope_theta = 2000,
                  eps = 1e-06, out_channels = NULL)
```

Arguments

in_channels	Integer. Packed latent channels (128).
num_layers	Integer. Double-stream block count (klein-4B: 5).
num_single_layers	Integer. Single-stream block count (20).
attention_head_dim	Integer. Per-head dimension.
num_attention_heads	Integer. Attention heads.
joint_attention_dim	Integer. Text embedding dim (7680).
mlp_ratio	Numeric. Feed-forward multiplier (3.0).
timestep_guidance_channels	Integer. Sinusoid width (256).
axes_dims_rope	Integer vector. Per-axis rotary dims.
rope_theta	Numeric. Rotary base frequency (2000).
eps	Numeric. Norm epsilon.
out_channels	Integer or NULL. Defaults to in_channels.

Value

Module whose forward(hidden_states, encoder_hidden_states, timestep, image_rotary_emb) returns the predicted velocity for the image tokens [B, S_img, out_channels]. timestep is in sigma space (0-1); it is scaled by 1000 internally.

---

flux2_unpack_latents_with_ids

Unpack FLUX.2 tokens back to a latent grid via position ids

Description

Scatters tokens to (H, W) positions taken from the id columns (H = column 2, W = column 3, 0-based values). Reference: Flux2KleinPipeline._unpack_latents_with_ids.

Usage

```
flux2_unpack_latents_with_ids(x, ids, height, width)
```

Arguments

x	Tensor [B, S, C] of tokens.
ids	Tensor [S, 4] (or [B, S, 4]) of position ids.
height, width	Integers. Packed grid dimensions.

Value

Tensor [B, C, height, width].

`flux2_unpatchify_latents` *Unpatchify FLUX.2 latents (channels -> 2x2)*

Description

Inverse of `flux2_patchify_latents`. Reference: Flux2KleinPipeline._unpatchify_latents.

Usage

```
flux2_unpatchify_latents(latents)
```

Arguments

`latents`	Tensor [B, 4C, H, W].

Value

Tensor [B, C, H * 2, W * 2].

`flux2_vae_decoder` *FLUX.2 VAE decoder*

Description

The AutoencoderKLFlux2 decode path: post_quant_conv (1x1, 32 channels) followed by the standard AutoencoderKL decoder body (reused from `vae_decoder_native`), plus the BatchNorm running statistics used for latent (de)normalization. Reference: src/diffusers/models/autoencoders/autoencoder_kl_flux2.py.

Usage

```
flux2_vae_decoder(latent_channels = 32L,
                  block_channels = c(512L, 512L, 256L, 128L), norm_groups = 32L)
```

Arguments

`latent_channels`	Integer (32 for FLUX.2).
`block_channels`	Decoder block channels (reversed encoder block_out_channels).
`norm_groups`	Integer. Group norm groups.

Value

Module whose forward(z) decodes [B, 32, H, W] latents to [B, 3, 8H, 8W] images; `$bn$running_mean` / `$bn$running_var` carry the normalization statistics.

`flux_ada_layer_norm_continuous` *FLUX continuous adaLN (final norm)*

Description

Scale/shift conditioning of the final norm. Note the chunk order: scale first, then shift (the reverse of adaLN-Zero). Reference: diffusers AdaLayerNormContinuous as used by FLUX norm_out (elementwise_affine = FALSE, eps = 1e-6).

Usage

```
flux_ada_layer_norm_continuous(dim, cond_dim = dim, bias = TRUE)
```

Arguments

`dim`	Integer. Model dimension.
`cond_dim`	Integer. Conditioning embedding dimension.
`bias`	Logical. Bias on the projection (TRUE for FLUX.1, FALSE for FLUX.2).

Value

Module whose forward(x, cond) returns `x` normalized and then scaled and shifted by the conditioning embedding, a tensor of the same shape as `x`.

`flux_ada_layer_norm_zero` *FLUX adaLN-Zero modulation (double-stream)*

Description

Projects the conditioning embedding to six modulation vectors and returns the msa-modulated input plus the remaining parameters. Reference: diffusers AdaLayerNormZero.

Usage

```
flux_ada_layer_norm_zero(dim)
```

Arguments

`dim`	Integer. Model dimension.

Value

Module whose forward(x, emb) returns `list(x_norm, gate_msa, shift_mlp, scale_mlp, gate_mlp)`.

flux_ada_layer_norm_zero_single *FLUX adaLN-Zero modulation (single-stream)*

Description

Three modulation vectors: shift, scale, gate. Reference: diffusers AdaLayerNormZeroSingle.

Usage

```
flux_ada_layer_norm_zero_single(dim)
```

Arguments

dim	Integer. Model dimension.

Value

Module whose forward(x, emb) returns `list(x_norm, gate)`.

flux_apply_rotary_emb *Apply FLUX rotary embeddings to a per-head tensor*

Description

Rotates adjacent element pairs of the last dimension: `out = x * cos + rotate_half(x) * sin` with pairs interleaved (elements 1,2 form the first complex pair). Math in float32, result cast back to the input dtype. Reference: apply_rotary_emb with use_real_unbind_dim = -1.

Usage

```
flux_apply_rotary_emb(x, freqs)
```

Arguments

x	Tensor of shape [B, H, S, D] (per-head layout).
freqs	List of two tensors (cos, sin), each [S, D], from `flux_pos_embed`.

Value

Tensor with the same shape and dtype as `x`.

flux_attention *FLUX joint attention*

Description

Multi-head attention with per-head RMS q/k norms and rotary position embeddings. With `added_kv = TRUE` (double-stream blocks) the text stream gets its own q/k/v projections and both streams attend jointly (text tokens first); the outputs are split back and projected per stream. With `pre_only = TRUE` (single-stream blocks) there is no output projection. Reference: diffusers FluxAttention + FluxAttnProcessor.

Usage

```
flux_attention(query_dim, heads, dim_head, added_kv = FALSE, pre_only = FALSE,
               eps = 1e-06, bias = TRUE)
```

Arguments

`query_dim`	Integer. Model dimension.
`heads`	Integer. Number of attention heads.
`dim_head`	Integer. Per-head dimension.
`added_kv`	Logical. Add text-stream projections (double blocks).
`pre_only`	Logical. Skip the output projection (single blocks).
`eps`	Numeric. RMS norm epsilon.
`bias`	Logical. Bias on the linear projections (TRUE for FLUX.1, FALSE for FLUX.2).

Value

Module whose forward(hidden_states, encoder_hidden_states, image_rotary_emb, chunk_size) returns the attended image stream [B, S, query_dim]. When `encoder_hidden_states` is supplied (double-stream blocks) it returns `list(image, text)` instead, each projected by its own output layer.

flux_double_block *FLUX double-stream (MMDiT) transformer block*

Description

Image and text streams each get adaLN-Zero modulation and a feed-forward; attention is joint across both streams. Reference: diffusers FluxTransformerBlock.

Usage

```
flux_double_block(dim, num_attention_heads, attention_head_dim)
```

Arguments

`dim`	Integer. Model dimension.
`num_attention_heads`	Integer. Attention heads.
`attention_head_dim`	Integer. Per-head dimension.

Value

Module whose forward(hidden_states, encoder_hidden_states, temb, image_rotary_emb) returns `list(encoder_hidden_states, hidden_states)`.

`flux_is_quant_key` *Test whether a FLUX key is in the quantization cast set*

Description

Test whether a FLUX key is in the quantization cast set

Usage

```
flux_is_quant_key(key)
```

Arguments

`key`	Character vector of parameter names (diffusers-style).

Value

Logical vector.

`flux_load_pipeline` *Load the FLUX.1-schnell pipeline*

Description

Loads the quantized transformer artifact plus the VAE decoder, CLIP and T5 text encoders, tokenizer, and scheduler config (from the HuggingFace cache populated by `download_flux1`). Components load to the CPU when `phase_offload` is on and move to the GPU only for their phase of the generation.

Usage

```
flux_load_pipeline(model_dir = NULL, device = "cuda", precision = NULL,
                   text_device = NULL, attn_chunk = NULL, phase_offload = TRUE,
                   pin = NULL, verbose = TRUE)
```

Arguments

model_dir	Quantized artifact directory (default: the download_flux1 location for precision), or a raw diffusers transformer directory for full-precision loading.
device	Character. Compute device.
precision	"nf4" or "fp8"; NULL picks the flux_memory_profile recommendation.
text_device	Where the text encoders compute. NULL (default) takes the flux_memory_profile recommendation: "cuda" on 14 GB+ cards (T5-XXL onloads in bfloat16 for its encode) and "cpu" below that (T5-XXL runs float32 in place, since its ~9.8 GB GPU encode phase would not fit).
attn_chunk	Integer or NULL. Attention query-chunk override.
phase_offload	Logical. One GPU tenant per phase.
pin	Logical or NULL. Page-lock the phase-swapped weights for DMA-rate transfer (see staging). NULL (default) resolves via options(diffuseR.pin_staging) then the host-RAM-aware recommend decision.
verbose	Logical.

Value

A flux_pipeline list.

---

flux_load_transformer *Load a FLUX transformer from any checkpoint format*

Description

Builds flux_transformer from the checkpoint's embedded config and loads the weights. Dispatches on the checkpoint format:

Usage

```
flux_load_transformer(ckpt, device = "cuda", dtype = "bfloat16", pin = TRUE,
                      fp8_resident = FALSE, verbose = TRUE, ...)
```

Arguments

ckpt	A checkpoint from flux_open_checkpoint or flux_open_quantized.
device	Character. Compute device.
dtype	Character. Model dtype ("bfloat16" or "float32"). For quantized formats this sets the resident (non-quantized) tensors and must match the compute dtype: bfloat16 for GPU compute, float32 for CPU compute.
pin	Logical. Pin fp8 host memory for faster transfers (streamed fp8 only).
fp8_resident	Logical. Keep the fp8 weights on device instead of streaming from the CPU - right for models whose whole quantized footprint fits in VRAM (FLUX.2 klein-4B: ~4 GB).
verbose	Logical.
...	Overrides for flux_transformer arguments (tiny test configs).

Details

- full precision (flux_open_checkpoint): weights stream into the model in dtype on device.
- "nf4" (flux_open_quantized): cast-set linears become ltx23_nf4_linear; the whole model (packed weights included) moves to device and stays resident.
- "fp8": cast-set linears become ltx23_fp8_linear; fp8 weights stay CPU-resident (optionally pinned) and stream to device inside each forward.

Value

The loaded flux_transformer in eval mode.

---

flux_memory_profile *Resolve a FLUX memory profile*

Description

A thin adapter over recommend for the FLUX.1 pipeline, kept for back-compatibility. recommend("flux1") is the policy; this reshapes it into the legacy profile fields the loader consumes. Precision now rises with VRAM (nf4 default, fp8 GPU-resident on 14 GB+ cards when safetensors can read float8, bf16 on 24 GB+); the old bands, which put fp8 in a narrow low-VRAM slot it can no longer fit, were backwards.

Usage

```
flux_memory_profile(vram_gb = NULL)
```

Arguments

vram_gb	Numeric or NULL. Available VRAM; auto-detected when NULL (via nvidia-smi).

Value

List with name, precision ("nf4"/"fp8"/"bf16"), attn_chunk, text_device, phase_offload, max_pixels, and (advisory) fork_suggested and note.

flux_open_checkpoint *Open a FLUX transformer checkpoint directory*

Description

Opens a diffusers-layout transformer directory lazily (headers only). Sharded checkpoints are resolved through the index.json weight map; single-file checkpoints are opened directly. The transformer `config.json` is attached as `$config`.

Usage

```
flux_open_checkpoint(transformer_dir)
```

Arguments

`transformer_dir`	Directory containing `config.json` and the `diffusion_pytorch_model*.safetensors` file(s).

Value

An object of class `ltx23_checkpoint` (shared checkpoint interface): list with `handle$get_tensor`, `keys`, `config`, and `path`.

flux_open_quantized *Open a quantized FLUX artifact directory*

Description

Opens the sharded NF4/fp8 artifact written by `flux_quantize` through the shared checkpoint interface. The manifest's embedded transformer config and `format` ride along, so `flux_load_transformer` needs nothing else.

Usage

```
flux_open_quantized(dir)
```

Arguments

`dir`	The quantized artifact directory (with manifest.json).

Value

An `ltx23_checkpoint` with `$format` set.

flux_pack_latents *Pack FLUX latents into a patch sequence*

Description

Packs a [B, C, H, W] latent into 2x2 patches, giving a sequence [B, (H/2) * (W/2), C * 4]. Reference: FluxPipeline._pack_latents.

Usage

```
flux_pack_latents(latents)
```

Arguments

latents	Tensor of shape [B, C, H, W]; H and W must be even.

Value

Tensor of shape [B, (H/2) * (W/2), C * 4].

flux_pos_embed *Compute FLUX rotary frequencies from position ids*

Description

Per-axis 1D rotary frequencies (interleaved-real convention), computed in float64 on CPU and concatenated over the axes. Reference: FluxPosEmbed with get_1d_rotary_pos_embed(repeat_interleave_real=TRUE, use_real=TRUE, freqs_dtype=float64).

Usage

```
flux_pos_embed(ids, axes_dim = c(16L, 56L, 56L), theta = 10000)
```

Arguments

ids	Tensor of shape [S, 3]: concatenated text ids (all zero) and image ids from flux_prepare_latent_image_ids.
axes_dim	Integer vector of per-axis rotary dims; must sum to the attention head dim. FLUX uses c(16, 56, 56).
theta	Numeric. RoPE base frequency.

Value

List of two tensors (cos, sin), each [S, sum(axes_dim)], float32, on the device of ids.

flux_prepare_latent_image_ids *Build FLUX latent image position ids*

Description

Position ids over the packed latent grid (latent height/2 x width/2). Channel 1 is always zero, channel 2 holds the row index, channel 3 the column index. Reference: FluxPipeline._prepare_latent_image_ids.

Usage

```
flux_prepare_latent_image_ids(height, width, device = "cpu")
```

Arguments

height	Integer. Packed grid height (latent height / 2).
width	Integer. Packed grid width (latent width / 2).
device	Device for the resulting tensor.

Value

Float tensor of shape [height * width, 3].

flux_quantize *Quantize a FLUX transformer to NF4 or fp8 shards*

Description

Streams the bf16 diffusers checkpoint tensor by tensor. Cast-set weights (see flux_is_quant_key) are stored as NF4 (packed uint8 + <key>_absmax float32 blocks) or as float8_e4m3fn with an absmax/448 per-tensor <key>_scale; everything else is copied through unchanged. The manifest embeds the transformer config, so the source checkpoint is not needed again after quantization.

Usage

```
flux_quantize(transformer_dir, output_dir = NULL, format = c("nf4", "fp8"),
              shard_bytes = 1.9e+09, force = FALSE, verbose = TRUE)
```

Arguments

transformer_dir	Source diffusers transformer directory.
output_dir	Output directory for shards + manifest (default: the per-format location under tools::R_user_dir).
format	"nf4" or "fp8".
shard_bytes	Numeric. Target shard size in bytes. The default 1.9e9 keeps every shard under the 2^31-byte (~2.15 GB) ceiling that stock CRAN safetensors can read, so the artifact loads fork-free. Pass a larger value (e.g. 4e9) only for local builds you will read back with a fork-patched safetensors.
force	Logical. Re-quantize even if a valid manifest exists.
verbose	Logical.

Value

Invisibly, the manifest list.

---

flux_single_block *FLUX single-stream transformer block*

Description

Parallel attention + MLP over the joint [text; image] sequence with a shared gate: `x + gate * proj_out(cat(attn, gelu(mlp)))`. The reference concatenates the streams inside every block and splits after; here the caller concatenates once before the single-block stack, which is numerically identical. Reference: diffusers FluxSingleTransformerBlock.

Usage

```
flux_single_block(dim, num_attention_heads, attention_head_dim, mlp_ratio = 4)
```

Arguments

dim	Integer. Model dimension.
num_attention_heads	Integer. Attention heads.
attention_head_dim	Integer. Per-head dimension.
mlp_ratio	Numeric. MLP hidden dim multiplier.

Value

Module whose forward(hidden_states, temb, image_rotary_emb) returns the joint hidden states.

`flux_transformer` *FLUX transformer model*

Description

19 double-stream (MMDiT) blocks followed by 38 single-stream blocks over the joint [text; image] sequence, with adaLN-Zero conditioning on timestep + pooled CLIP text. Rotary embeddings are precomputed by the caller with `flux_pos_embed` (they are static across denoise steps). Defaults are the FLUX.1-schnell configuration.

Usage

```
flux_transformer(in_channels = 64L, num_layers = 19L, num_single_layers = 38L,
                 attention_head_dim = 128L, num_attention_heads = 24L,
                 joint_attention_dim = 4096L, pooled_projection_dim = 768L,
                 axes_dims_rope = c(16L, 56L, 56L), out_channels = NULL)
```

Arguments

`in_channels`	Integer. Packed latent channels (64).
`num_layers`	Integer. Double-stream block count.
`num_single_layers`	Integer. Single-stream block count.
`attention_head_dim`	Integer. Per-head dimension.
`num_attention_heads`	Integer. Attention heads.
`joint_attention_dim`	Integer. T5 embedding dim (4096).
`pooled_projection_dim`	Integer. CLIP pooled dim (768).
`axes_dims_rope`	Integer vector. Per-axis rotary dims.
`out_channels`	Integer or NULL. Output channels (defaults to `in_channels`).

Value

Module whose forward(hidden_states, encoder_hidden_states, pooled_projections, timestep, image_rotary_emb) returns the predicted velocity for the image tokens [B, S_img, out_channels]. `timestep` is in sigma space (0-1); it is scaled by 1000 internally, matching the reference.

flux_unpack_latents *Unpack a FLUX patch sequence back into latents*

Description

Inverse of `flux_pack_latents`. Height and width are the target image dimensions in pixels; the latent grid is derived via the VAE scale factor and the 2x2 patch size. Reference: FluxPipeline._unpack_latents.

Usage

```
flux_unpack_latents(latents, height, width, vae_scale_factor = 8L)
```

Arguments

`latents`	Tensor of shape [B, S, C_packed].
`vae_scale_factor`	Integer. Spatial downsampling of the VAE (8).
`height, width`	Integers. Image height/width in pixels.

Value

Tensor of shape [B, C_packed / 4, height / 8, width / 8].

fp8_ltx23 *FP8 Weight Storage for the LTX-2.3 Transformer*

Description

GPU-poor weight handling: the large attention/FFN linears of the DiT are stored as float8_e4m3fn with per-tensor scales (the official LTX quantization policy), kept CPU-resident (optionally pinned), and dequantized on the compute device inside each forward. Everything else (norms, embeddings, modulation tables, biases) stays bfloat16. Requires a safetensors build with F8 dtype support.

gemma3_config_ltx2 *Create Gemma3 configuration for LTX-2*

Description

Returns the default configuration used by LTX-2's text encoder.

Usage

```
gemma3_config_ltx2()
```

Value

List with model configuration parameters.

gemma3_encode_batch *Batch-encode prompts with Gemma3, cached to disk*

Description

Encodes a vector of prompts in sub-batches (bounding activation VRAM) and optionally caches each prompt's result under `cache_dir`, keyed by the prompt text and sequence length. Already-cached prompts are skipped, so an interrupted batch resumes where it stopped. Embeddings land on the CPU either way; the renderer moves them per phase.

Usage

```
gemma3_encode_batch(prompts, model = NULL, tokenizer = NULL, batch_size = 4L,
                    cache_dir = NULL, max_sequence_length = 1024L,
                    device = "cuda", verbose = TRUE)
```

Arguments

`prompts`	Character vector.
`batch_size`	Integer. Prompts per forward pass (default 4; raise on cards with headroom, lower if the encode OOMs).
`cache_dir`	Optional directory. When given, each prompt is written to `gemma3-<md5>.pt` and the return value is the character vector of those paths (load one with `torch::torch_load`); when NULL, the return value is a list of `list(prompt_embeds, prompt_attention_mask)` in prompt order.
`model`, `tokenizer`	As in `encode_with_gemma3`.
`max_sequence_length`, `device`, `verbose`	As in `encode_with_gemma3`.

Details

Budget note: each prompt's result is the full hidden-state stack the LTX connectors consume - roughly 0.4 GB at 1024 tokens - so caching N prompts needs ~0.4N GB of disk.

Value

Character vector of cache paths (with `cache_dir`) or a list of per-prompt embedding results (without).

`gemma3_quantize_nf4` *Quantize a Gemma3 text encoder to NF4 shards*

Description

Streams the HuggingFace Gemma3 checkpoint tensor by tensor. The language model's projection weights (q/k/v/o and gate/up/down, ~11B of the 12B parameters) are stored as NF4 (packed uint8 + <key>_absmax float32 blocks); embeddings and norms are copied at the resident dtype. Vision-tower and projector weights are dropped - the text encoder never uses them. The result is a ~8 GB artifact that fits a 16 GB card during the encode phase (vs 45 GB of host RAM for the fp32 CPU path).

Usage

```
gemma3_quantize_nf4(model_path, output_dir = NULL, shard_bytes = 1.9e+09,
                    force = FALSE, verbose = TRUE)
```

Arguments

`model_path`	HuggingFace snapshot directory (config.json + model-*.safetensors).
`output_dir`	Output directory for shards + manifest (default: `gemma3-nf4` under `tools::R_user_dir`).
`shard_bytes`	Numeric. Target shard size in bytes; the 1.9e9 default keeps shards readable by stock CRAN safetensors.
`force`	Logical. Re-quantize even if a valid manifest exists.
`verbose`	Logical.

Details

Keys are stored normalized (`language_model.` / `model.` prefixes stripped), matching the module tree of `gemma3_text_model`.

Value

Invisibly, the manifest list.

gemma3_text_model *Gemma3 Text Model*

Description

Full Gemma3 text encoder model.

Usage

```
gemma3_text_model(config)
```

Arguments

config	Model configuration list.

Value

Module whose forward(input_ids, ...) returns `list(last_hidden_state, hidden_states)`: the final hidden state [B, S, hidden_size] and the list of per-layer hidden states.

gemma3_tokenizer *Gemma3 Tokenizer*

Description

Native R tokenizer for Gemma3 using BPE. Loads from HuggingFace tokenizer.json format.

Usage

```
gemma3_tokenizer(tokenizer_path)
```

Arguments

tokenizer_path	Character. Path to tokenizer directory or tokenizer.json file.

Value

A gemma3_tokenizer object (extends bpe_tokenizer).

get_required_components *Get required components for each model type*

Description

This function returns a list of required components for each supported model type.

Usage

```
get_required_components(model_name)
```

Arguments

model_name	A character string representing the name of the model.

Value

A character vector of required components for the specified model.

img2img *Image-to-Image Generation with Stable Diffusion*

Description

This function generates an image based on an input image and a text prompt using the Stable Diffusion model. It allows for various configurations such as model name, device, scheduler, and more.

Usage

```
img2img(input_image, prompt, negative_prompt = NULL, img_dim = 512,
        model_name = c("sd21", "sdxl"), pipeline = NULL, devices = "auto",
        unet_dtype_str = "float16", download_models = FALSE,
        scheduler = "ddim", num_inference_steps = 50, strength = 0.8,
        guidance_scale = 7.5, seed = NULL, save_file = TRUE, filename = NULL,
        metadata_path = NULL, use_native_decoder = FALSE,
        use_native_text_encoder = FALSE, use_native_unet = FALSE, ...)
```

Arguments

input_image	Path to the input image or a tensor representing the image.
prompt	Text prompt to guide the image generation.
negative_prompt	Optional negative prompt to guide the image generation.‘
img_dim	Dimension of the output image (default: 512).
model_name	Name of the Stable Diffusion model to use (default: "sd21").
pipeline	Optional pre-loaded pipeline. If ‘NULL‘, it will be loaded based on ‘model_name‘.
devices	A named list of devices for each model component (e.g., ‘list(unet = "cuda", decoder = "cpu", text_encoder = "cpu", encoder = "cpu")‘).
unet_dtype_str	Optional A character for dtype of the unet component (typically "torch_float16" for cuda and "torch_float32" for cpu).
download_models	Logical indicating whether to download models if not found (default: FALSE).
scheduler	Scheduler to use for the diffusion process (default: "ddim").
num_inference_steps	Number of diffusion steps (default: 50).
strength	Strength of the image-to-image transformation (default: 0.8).
guidance_scale	Scale for classifier-free guidance (default: 7.5).
seed	Random seed for reproducibility (default: NULL).
save_file	Logical indicating whether to save the generated image.
filename	Optional filename for saving the image. If ‘NULL‘, a default name is generated.
metadata_path	Path to save metadata (default: NULL).
use_native_decoder	Logical; if TRUE, uses native R torch decoder instead of TorchScript. Native decoder has better GPU compatibility (especially Blackwell).
use_native_text_encoder	Logical; if TRUE, uses native R torch text encoder instead of TorchScript. Native text encoder has better GPU compatibility (especially Blackwell).
use_native_unet	Logical; if TRUE, uses native R torch UNet instead of TorchScript. Native UNet has better GPU compatibility (especially Blackwell).
...	Additional arguments for future use.

Value

An image array and metadata

is_blackwell_gpu *Check if GPU is Blackwell Architecture*

Description

Blackwell GPUs (RTX 50xx) may need special handling.

Usage

```
is_blackwell_gpu()
```

Value

Logical. TRUE if Blackwell GPU detected.

Examples

```
# Soft probe: FALSE on any machine without a Blackwell card, and on
# machines where torch has no lantern binaries.
is_blackwell_gpu()
```

jit_ltx23 *LTX-2.3 JIT Block Stack*

Description

TorchScript compilation of the 48-block NF4 transformer step (cf. the torch skill's JIT-decode pattern proven in whisper and chatterbox). Eager execution crosses R -> lantern per op (~190 us each) and leaves every intermediate as an R tensor handle that only dies at gc(); at high resolution that forces a per-block gc() costing the vast majority of step time. Compiled, the whole block stack is one crossing: intermediates are freed eagerly by libtorch, no R garbage accumulates, no per-block gc is needed, and attention runs through the fused `scaled_dot_product_attention` kernel instead of a materialized score matrix.

Details

Weights are passed per call as a flat `List[Tensor]` (borrowed by reference, no copies) with a fixed per-block layout; the packer and the TorchScript indices must stay in lockstep (parity-tested).

jit_vae_ltx23 *JIT-Traced Decode for the LTX-2.3 VAEs and Vocoder*

Description

The video/audio decoders and the vocoder are static feed-forward graphs, so `torch::jit_trace` converts them wholesale: one R-to-libtorch crossing per forward, intermediates freed eagerly by libtorch instead of accumulating as R handles until gc. Traces are shape-specialized (runtime sizes bake into the graph as constants; a mismatched input errors), so they are cached per instance, input shape, dtype, device, and call tag, and re-traced on a miss.

Details

A trace captures the module's weight tensors, which would pin them on the GPU across phase offloads; the pipeline releases all traces whenever a component offloads (`.ltx23_release_vae_traces`).

Tracing hazard on this lantern build: if the allocator callback runs R's gc *during* trace recording (memory pressure), the recorded graph can capture garbage argument values (observed as corrupted `narrow` starts on the full-size decoder; verified 5/5 clean once gc cannot fire mid-trace). Defenses, in order: a gc + cache flush right before each trace so pressure starts near zero, `tryCatch` around trace and replay, and a one-time validation of every fresh trace against the eager output — any mismatch permanently blacklists that shape and runs eager. With those in place the traced path cannot corrupt output — but per render it measured slower than eager (traces are released on phase offload, so every render re-pays trace + validation), so it stays opt-in: `options(diffuseR.jit_vae = TRUE)`.

latents_to_video *Create Video from Latents (Helper)*

Description

Convenience function to decode latents and save video in one step.

Usage

```
latents_to_video(latents, vae, file, fps = 24, ...)
```

Arguments

`latents`	Tensor of latents from generation.
`vae`	VAE decoder module.
`file`	Output file path.
`fps`	Frames per second.
`...`	Additional arguments to save_video.

Value

Invisibly returns the output file path.

`load_decoder_safetensors` *Load HF safetensors VAE weights into the native decoder*

Description

Loads the decoder half of a diffusers AutoencoderKL safetensors file (e.g. FLUX.1-schnell's `vae/diffusion_pytorch_model.safetensors`). Keys under `decoder.` map to the native module 1:1; encoder and quant-conv keys are skipped (the FLUX VAE has no quant convs, and txt2img needs no encoder).

Usage

```
load_decoder_safetensors(native_decoder, path, verbose = TRUE)
```

Arguments

`native_decoder`	Native VAE decoder module
`path`	Path to the VAE .safetensors file (or a directory containing diffusion_pytorch_model.safetensors)
`verbose`	Print loading progress

Value

The native decoder with loaded weights (invisibly)

`load_decoder_weights` *Load weights from TorchScript decoder into native decoder*

Description

Load weights from TorchScript decoder into native decoder

Usage

```
load_decoder_weights(native_decoder, torchscript_path, verbose = TRUE)
```

Arguments

`native_decoder`	Native VAE decoder module
`torchscript_path`	Path to TorchScript decoder .pt file
`verbose`	Print loading progress

Value

The native decoder with loaded weights (invisibly)

`load_flux2_vae_decoder` *Load the FLUX.2 VAE decoder from safetensors*

Description

Loads the decoder half plus post_quant_conv and the BatchNorm running statistics; encoder and quant_conv keys are skipped (txt2img needs no encoder).

Usage

```
load_flux2_vae_decoder(path, latent_channels = 32L,
                       block_channels = c(512L, 512L, 256L, 128L),
                       norm_groups = 32L, verbose = TRUE)
```

Arguments

`path`	Path to the VAE .safetensors file (or a directory containing diffusion_pytorch_model.safetensors).
`verbose`	Logical.
`latent_channels, block_channels, norm_groups`	Constructor arguments for `flux2_vae_decoder`.

Value

The loaded `flux2_vae_decoder` in eval mode.

`load_gemma3_nf4` *Load a Gemma3 text encoder from an NF4 artifact*

Description

Builds the model as a skeleton at the compute dtype, swaps the projection linears for NF4 modules filled from the artifact (dequantized per forward through the shared byte-LUT), copies the residents, and hard-errors on any parameter the artifact does not fill.

Usage

```
load_gemma3_nf4(artifact_dir, device = "cuda", dtype = "bfloat16",
                pin = getOption("diffuseR.pin_staging", TRUE), verbose = TRUE)
```

Arguments

artifact_dir	Directory produced by gemma3_quantize_nf4.
device	"cuda" or "cpu".
dtype	Compute dtype ("bfloat16" default).
pin	Logical. When loading to the CPU, page-lock the weights so encode_with_gemma3 can swap the model to the GPU at DMA speed per encode and back for free (see staging). Default follows options(diffuseR.pin_staging).
verbose	Logical.

Value

A gemma3_text_model ready for encode_with_gemma3.

---

load_gemma3_text_encoder

Load Gemma3 Text Model from safetensors

Description

Loads pre-trained Gemma3 weights from HuggingFace safetensors files. An NF4 artifact directory (from gemma3_quantize_nf4) dispatches to load_gemma3_nf4.

Usage

```
load_gemma3_text_encoder(model_path, device = "cpu", dtype = "float16",
                         pin = getOption("diffuseR.pin_staging", TRUE),
                         verbose = TRUE)
```

Arguments

model_path	Character. Path to directory containing model files.
device	Character. Device to load model to.
dtype	Character. Data type ("float32", "float16", "bfloat16").
pin	Logical. When loading to the CPU, page-lock the weights so encode_with_gemma3 can swap the model to the GPU at DMA speed per encode (~0.3 s on, free off) instead of holding VRAM or reloading. Default follows options(diffuseR.pin_staging).
verbose	Logical. Print loading progress.

Value

Initialized gemma3_text_model with loaded weights.

`load_model_component` *Load a specific component of a diffusion model*

Description

Loads a TorchScript model component (UNet, decoder, or text encoder) from the hfhub cache or legacy model directory, downloading it first if necessary.

Usage

```
load_model_component(component, model_name = "sd21", device = "cpu",
                     unet_dtype_str = NULL, download = TRUE, use_native = FALSE)
```

Arguments

`component`	Character string, the component to load: "unet", "decoder", or "text_encoder".
`model_name`	Character string, the name of the model to use.
`device`	Character string, the torch device to load the model onto ("cpu" or "cuda").
`unet_dtype_str`	Optional; the data type for the UNet model. If 'NULL', defaults to 'float32' for CPU and 'float16' for CUDA.
`download`	Logical; if 'TRUE' (default), downloads the model if it doesn't exist locally.
`use_native`	Logical; if 'TRUE', uses native R torch modules instead of TorchScript. Supported for unet, decoder, text_encoder, and text_encoder2. Native modules have better GPU compatibility (especially on Blackwell/RTX 50xx).

Value

A torch model object.

Examples

```
## Not run:
unet <- load_model_component("unet", "sd21", "cpu")

## End(Not run)
```

load_pipeline *Load a diffusion model pipeline*

Description

This function loads a diffusion model pipeline consisting of a UNet, VAE decoder, and text encoder. It initializes the models and sets up the environment for inference.

Usage

```
load_pipeline(model_name, m2d, i2i = FALSE, unet_dtype_str,
              use_native_decoder = FALSE, use_native_text_encoder = FALSE,
              use_native_unet = FALSE, ...)
```

Arguments

model_name	The name of the model to load.
m2d	A list containing model-to-device mappings and configurations.
i2i	Logical indicating whether to load the encoder for img2img().
unet_dtype_str	A string representing the data type for the UNet model (e.g., "float32", "float16").
use_native_decoder	Logical; if TRUE, uses native R torch decoder instead of TorchScript. Native decoder has better GPU compatibility (especially Blackwell).
use_native_text_encoder	Logical; if TRUE, uses native R torch text encoder instead of TorchScript. Native text encoder has better GPU compatibility (especially Blackwell).
use_native_unet	Logical; if TRUE, uses native R torch UNet instead of TorchScript. Native UNet has better GPU compatibility (especially Blackwell).
...	Additional arguments passed to the model loading functions.

Value

An environment containing the loaded models and configuration.

Examples

```
## Not run:
pipeline <- load_pipeline("my_model", device = "cuda")

## End(Not run)
```

load_qwen3_text_encoder

Load a Qwen3 encoder from a transformers directory

Description

Streams the (possibly sharded) safetensors weights into `qwen3_encoder`. The LM head is tied to the embeddings and skipped.

Usage

```
load_qwen3_text_encoder(model_path, device = "cpu", dtype = "float32",
                        verbose = TRUE, ...)
```

Arguments

`model_path`	Directory with `config.json` and `model*.safetensors` (FLUX.2-klein's `text_encoder`).
`device`	Character. Target device.
`dtype`	Character. "bfloat16" (GPU) or "float32" (CPU).
`verbose`	Logical.
`...`	Overrides for `qwen3_encoder` arguments.

Value

The loaded `qwen3_encoder` in eval mode.

load_t5_text_encoder

Load a T5 encoder from a transformers directory

Description

Streams the (possibly sharded) safetensors weights into `t5_encoder`, stripping the `encoder.` key prefix and aliasing `embed_tokens` to the shared embedding.

Usage

```
load_t5_text_encoder(model_path, device = "cpu", dtype = "float32",
                     verbose = TRUE, ...)
```

Arguments

`model_path`	Directory with `config.json` and `model*.safetensors` (FLUX.1-schnell's `text_encoder_2`).
`device`	Character. Target device.
`dtype`	Character. "float32" (CPU default; T5 overflows in float16) or "bfloat16".
`verbose`	Logical.
`...`	Overrides for `t5_encoder` arguments.

Value

The loaded `t5_encoder` in eval mode.

`load_text_encoder2_safetensors`
Load HF safetensors weights into the native SDXL text encoder 2

Description

Loads a HuggingFace CLIPTextModelWithProjection `model.safetensors` (SDXL's OpenCLIP ViT-bigG `text_encoder_2`) into `text_encoder2_native`. Same key layout as encoder 1, plus the top-level `text_projection.weight` that produces the pooled `text_embeds`.

Usage

```
load_text_encoder2_safetensors(native_encoder, path, verbose = TRUE)
```

Arguments

`native_encoder`	Native text encoder 2 module
`path`	Path to model.safetensors (or a directory containing it)
`verbose`	Print loading progress

Value

The native encoder with loaded weights (invisibly)

`load_text_encoder2_weights`
Load weights from TorchScript text encoder 2 into native encoder

---

Description

Load weights from TorchScript text encoder 2 into native encoder

Usage

```
load_text_encoder2_weights(native_encoder, torchscript_path, verbose = TRUE)
```

Arguments

`native_encoder`	Native text encoder 2 module
`torchscript_path`	Path to TorchScript encoder .pt file
`verbose`	Print loading progress

Value

The native encoder with loaded weights (invisibly)

---

load_text_encoder_safetensors

Load HF safetensors weights into the native CLIP text encoder

---

Description

Loads a HuggingFace CLIPTextModel `model.safetensors` (e.g. FLUX.1-schnell's `text_encoder` or SDXL's `text_encoder`) into `text_encoder_native`, reusing the TorchScript key remaps minus the export prefixes.

Usage

```
load_text_encoder_safetensors(native_encoder, path, verbose = TRUE)
```

Arguments

`native_encoder`	Native text encoder module
`path`	Path to model.safetensors (or a directory containing it)
`verbose`	Print loading progress

Value

The native encoder with loaded weights (invisibly)

---

load_text_encoder_weights

Load weights from TorchScript text encoder into native encoder

---

Description

Load weights from TorchScript text encoder into native encoder

Usage

```
load_text_encoder_weights(native_encoder, torchscript_path, verbose = TRUE)
```

Arguments

`native_encoder`	Native text encoder module
`torchscript_path`	Path to TorchScript encoder .pt file
`verbose`	Print loading progress

Value

The native encoder with loaded weights (invisibly)

---

load_to_gpu *Load Module to GPU*

Description

Moves a torch module and all its parameters to CUDA.

Usage

```
load_to_gpu(module, device = "cuda")
```

Arguments

module	A torch nn_module.
device	Character. Target device (default "cuda").

Value

The module (modified in place).

Examples

```
if (torch::torch_is_installed()) {
  model <- torch::nn_linear(4, 2)
  # "cuda" needs a GPU; "cpu" is the portable round trip.
  load_to_gpu(model, device = "cpu")
}
```

---

load_unet_safetensors *Load HF safetensors weights into the native SD21 UNet*

Description

Load HF safetensors weights into the native SD21 UNet

Usage

```
load_unet_safetensors(native_unet, path, verbose = TRUE)
```

Arguments

native_unet	A unet_native module.
path	Path to the UNet directory (containing diffusion_pytorch_model.safetensors or its shard index) or directly to the single-file checkpoint.
verbose	Print how many parameters were loaded.

Value

The native UNet with weights loaded (invisibly).

---

load_unet_sdxl_safetensors

Load HF safetensors weights into the native SDXL UNet

Description

Load HF safetensors weights into the native SDXL UNet

Usage

```
load_unet_sdxl_safetensors(native_unet, path, verbose = TRUE)
```

Arguments

native_unet	A unet_sdxl_native module.
path	Path to the UNet directory (containing diffusion_pytorch_model.safetensors or its shard index) or directly to the single-file checkpoint.
verbose	Print how many parameters were loaded.

Value

The native UNet with weights loaded (invisibly).

load_unet_sdxl_weights *Load weights from TorchScript SDXL UNet into native SDXL UNet*

Description

Load weights from TorchScript SDXL UNet into native SDXL UNet

Usage

```
load_unet_sdxl_weights(native_unet, torchscript_path, verbose = TRUE)
```

Arguments

native_unet	Native SDXL UNet module
torchscript_path	Path to TorchScript SDXL UNet .pt file
verbose	Print loading progress

Value

The native UNet with loaded weights (invisibly)

load_unet_weights *Load weights from TorchScript UNet into native UNet*

Description

Load weights from TorchScript UNet into native UNet

Usage

```
load_unet_weights(native_unet, torchscript_path, verbose = TRUE)
```

Arguments

native_unet	Native UNet module
torchscript_path	Path to TorchScript UNet .pt file
verbose	Print loading progress

Value

The native UNet with loaded weights (invisibly)

ltx23_adain_filter_latent

Adaptive instance normalization between latent tensors

Description

Matches each (batch, channel) slice's mean/std to the reference latents, blended by `factor` (cf. diffusers `LTX2LatentUpsamplePipeline.adain_filter_latent`).

Usage

```
ltx23_adain_filter_latent(latents, reference_latents, factor = 1)
```

Arguments

`latents`	Tensor [B, C, F, H, W].
`reference_latents`	Tensor with the target statistics.
`factor`	Numeric blend in [-10, 10]; 0 is identity.

Value

Filtered latents.

ltx23_ada_layer_norm_single

Adaptive layer norm single (adaLN-single)

Description

Embeds a timestep/sigma and projects it to a configurable number of modulation parameter vectors.

Usage

```
ltx23_ada_layer_norm_single(embedding_dim, num_mod_params = 6L)
```

Arguments

`embedding_dim`	Integer. Model dimension.
`num_mod_params`	Integer. Number of modulation parameter vectors.

Value

Module whose forward returns `list(mod_params [N, num_mod_params * dim], embedded_timestep [N, dim])`.

ltx23_antialias_act1d *Anti-aliased activation*

Description

Upsample 2x, apply the activation, downsample 2x.

Usage

```
ltx23_antialias_act1d(channels, ratio = 2L, kernel_size = 12L)
```

Arguments

channels	Integer. Channels for the SnakeBeta activation.
ratio, kernel_size	Integers. Resampling config.

Value

Module whose forward(x) returns the activation applied at 2x rate (upsample, activate, downsample), a tensor of the same shape as x, with the aliasing the raw activation would introduce filtered out.

ltx23_apply_interleaved_rotary_emb *Apply interleaved rotary embeddings*

Description

Rotates adjacent element pairs of the last dimension: `out = x * cos + rotate_half(x) * sin` with pairs interleaved (elements 1,2 form the first complex pair).

Usage

```
ltx23_apply_interleaved_rotary_emb(x, freqs)
```

Arguments

x	Tensor of shape [B, S, C].
freqs	List of two tensors (cos, sin), each [B, S, C].

Value

Tensor with the same shape and dtype as x.

ltx23_apply_split_rotary_emb *Apply split rotary embeddings*

Description

Rotates element pairs formed by splitting the last dimension in half: element i pairs with element i + d/2. The cos/sin tensors carry half the head dimension.

Usage

```
ltx23_apply_split_rotary_emb(x, freqs)
```

Arguments

x	Tensor of shape [B, H, T, D] (per-head layout), or [B, T, H*D] which is reshaped per-head when freqs is 4D.
freqs	List of two tensors (cos, sin), each [B, H, T, D/2].

Value

Tensor with the same shape and dtype as x.

ltx23_attention *LTX-2 attention layer*

Description

Attention with RMS q/k norms across heads, optional per-head output gating (LTX-2.3), separate query/key RoPE (for a2v/v2a cross attention), and optional STG perturbation (skip attention, use the value projection).

Usage

```
ltx23_attention(query_dim, heads = 8L, kv_heads = NULL, dim_head = 64L,
                bias = TRUE, cross_attention_dim = NULL, out_bias = TRUE,
                norm_eps = 1e-06, norm_elementwise_affine = TRUE,
                rope_type = "split", apply_gated_attention = FALSE)
```

Arguments

query_dim	Integer. Query feature dimension.
dim_head	Integer. Per-head dimension.
cross_attention_dim	Integer or NULL. Key/value input dimension (NULL for self-attention).
norm_eps	Numeric. RMS norm epsilon.
norm_elementwise_affine	Logical. RMS norms carry weights.
rope_type	"split" or "interleaved".
apply_gated_attention	Logical. Add per-head sigmoid output gates.
heads, kv_heads	Integers. Attention head counts.
bias, out_bias	Logicals. Projection biases.

Value

Module whose forward(hidden_states, ...) returns the attended states [B, S, query_dim] after the output projection.

---

ltx23_audio_causal_conv2d

Causal 2D convolution for audio spectrograms

Description

Pads asymmetrically along the causal axis ("height" = time frames for LTX audio) before an unpadded Conv2d.

Usage

```
ltx23_audio_causal_conv2d(in_channels, out_channels, kernel_size = 3L,
                          stride = 1L, causality_axis = "height")
```

Arguments

kernel_size	Integer or length-2 vector.
stride	Integer.
causality_axis	"height", "width", "width-compatibility", or "none".
in_channels, out_channels	Integers.

Value

Module whose forward(x) returns the convolved tensor, padded so that each output frame depends only on current and earlier input frames.

ltx23_audio_decoder *LTX-2.3 audio VAE decoder*

Description

Latents [B, 8, L, 16] -> mel spectrogram [B, 2, 4L - 3, 64].

Usage

```
ltx23_audio_decoder(base_channels = 128L, output_channels = 2L,
                    num_res_blocks = 2L, latent_channels = 8L,
                    ch_mult = c(1L, 2L, 4L), causality_axis = "height",
                    mel_bins = 64L)
```

Arguments

base_channels	Integer.
output_channels	Integer. Audio channels (2 = stereo).
num_res_blocks	Integer. Per-level ResNet count (a stage runs num_res_blocks + 1 blocks).
latent_channels	Integer.
ch_mult	Integer vector. Channel multipliers per level.
causality_axis	Character.
mel_bins	Integer. Output mel bins (crop/pad target).

Value

Module whose forward(x) returns the decoded mel spectrogram reconstructed from an audio latent.

ltx23_audio_downsample *LTX audio downsampler*

Description

Causal zero-pad followed by a plain stride-2 conv (reference LTX2AudioDownsample; note the conv is unwrapped, so its checkpoint key is downsample.conv.*).

Usage

```
ltx23_audio_downsample(in_channels, causality_axis = "height")
```

Arguments

in_channels	Integer.
causality_axis	Character.

Value

Module whose forward(x) returns the strided convolution of x, halving the downsampled axes.

---

ltx23_audio_encoder *LTX-2.3 audio VAE encoder*

Description

Mel spectrogram [B, 2, T, 64] -> latent distribution moments [B, 2 * latent_channels, ceil(T/4), 16]. Structure mirrors the decoder: causal convs, parameterless pixel norms, ResNet stages with stride-2 downsampling between levels (reference LTX2AudioEncoder).

Usage

```
ltx23_audio_encoder(base_channels = 128L, in_channels = 2L,
                    num_res_blocks = 2L, latent_channels = 8L,
                    ch_mult = c(1L, 2L, 4L), causality_axis = "height")
```

Arguments

in_channels	Integer. Mel channels (2 = stereo).
base_channels, num_res_blocks, latent_channels, ch_mult, causality_axis	See ltx23_audio_decoder.

Value

Module whose forward(x) returns the encoded audio latent, a tensor downsampled along time and mel axes with the configured latent channel count.

ltx23_audio_mel_frontend

Build the 16 kHz log-mel frontend for audio conditioning

Description

An `ltx23_mel_stft` whose STFT and mel bases are constructed (not checkpoint-loaded) with the audio VAE's preprocessing spec.

Usage

```
ltx23_audio_mel_frontend(filter_length = 1024L, hop_length = 160L,
                         n_mels = 64L, sample_rate = 16000L, fmin = 0,
                         fmax = 8000)
```

Arguments

`filter_length`, `hop_length`, `n_mels`, `sample_rate`, `fmin`, `fmax`
: The checkpoint preprocessing parameters (defaults are LTX-2.3's).

Value

An `ltx23_mel_stft` module.

ltx23_audio_resnet_block

LTX audio ResNet block

Description

PixelNorm -> SiLU -> causal conv, twice, with a 1x1 causal conv shortcut (`nin_shortcut`) on channel change.

Usage

```
ltx23_audio_resnet_block(in_channels, out_channels = NULL,
                         causality_axis = "height")
```

Arguments

`causality_axis`
: Character.

`in_channels`, `out_channels`
: Integers.

Value

Module whose forward(x) returns `x` plus the residual branch, a tensor of the same shape as `x`.

ltx23_audio_upsample *LTX audio upsampler*

Description

Nearest 2x interpolation, causal conv, then a crop of the first element along the causal axis.

Usage

```
ltx23_audio_upsample(in_channels, causality_axis = "height")
```

Arguments

in_channels	Integer.
causality_axis	Character.

Value

Module whose forward(x) returns the tensor upsampled 2x by nearest-neighbour interpolation and convolved.

ltx23_audio_vae *LTX-2.3 audio VAE*

Description

Encoder + decoder plus the per-channel latent statistics loaded from the checkpoint. Encoding is used for audio-conditioned generation (lip sync); decoding for generated audio.

Usage

```
ltx23_audio_vae(base_channels = 128L, output_channels = 2L,
                num_res_blocks = 2L, latent_channels = 8L,
                ch_mult = c(1L, 2L, 4L), causality_axis = "height",
                mel_bins = 64L, in_channels = 2L)
```

Arguments

in_channels	Integer. Mel input channels (2 = stereo).
base_channels, output_channels, num_res_blocks, latent_channels, ch_mult, causality_axis, mel_bins	See ltx23_audio_decoder.

Value

Module bundling the audio encoder and decoder. Its forward(z) is decode(z), returning the mel spectrogram for a latent; $encode() and $decode() are callable separately.

ltx23_causal_conv3d *Causal 3D convolution*

Description

Spatial padding is handled by the convolution; temporal padding replicates the first frame (causal) or both edge frames (non-causal), chosen at call time.

Usage

```
ltx23_causal_conv3d(in_channels, out_channels, kernel_size = 3L, stride = 1L,
                    spatial_padding_mode = "zeros")
```

Arguments

kernel_size	Integer or length-3 vector (t, h, w).
stride	Integer or length-3 vector.
spatial_padding_mode	Character. Conv padding mode.
in_channels, out_channels	Integers.

Value

Module whose forward(hidden_states, causal) returns the 3-D convolution of the input. With causal = TRUE the temporal axis is left-padded by replicating the first frame, so no output frame sees a later input frame.

ltx23_census *Summarize checkpoint key coverage*

Description

Summarize checkpoint key coverage

Usage

```
ltx23_census(ckpt)
```

Arguments

ckpt	An ltx23_checkpoint.

Value

A data.frame with one row per component group and its key count.

ltx23_connector_transformer_1d

1D connector transformer

Description

Replaces padded positions with learnable registers (valid tokens are front-aligned in their original order; the tail is filled with registers indexed by absolute position, after which the attention mask is cleared), then runs 1D transformer blocks with rotary embeddings.

Usage

```
ltx23_connector_transformer_1d(num_attention_heads = 32L,
                               attention_head_dim = 128L, num_layers = 8L,
                               num_learnable_registers = 128L,
                               rope_base_seq_len = 4096L, rope_theta = 10000,
                               rope_double_precision = TRUE, eps = 1e-06,
                               rope_type = "split", gated_attention = TRUE)
```

Arguments

`num_learnable_registers`
: Integer or NULL. Register count (the sequence length must be divisible by it).

`eps`
: Numeric. Norm epsilon.

`gated_attention`
: Logical. Per-head attention output gates.

`num_attention_heads`, `attention_head_dim`, `num_layers`
: Transformer shape.

`rope_base_seq_len`, `rope_theta`, `rope_double_precision`, `rope_type`
: RoPE config.

Value

Module whose forward(hidden_states, attention_mask, attn_mask_binarize_threshold) returns `list(hidden_states, attention_mask)`: the transformed sequence and the (possibly binarized) mask that accompanies it.

ltx23_denormalize_latents

Denormalize latents with the VAE's per-channel statistics

Description

Denormalize latents with the VAE's per-channel statistics

Usage

```
ltx23_denormalize_latents(latents, latents_mean, latents_std)
```

Arguments

latents	Tensor [B, C, F, H, W].
latents_mean, latents_std	Tensors [C].

Value

Denormalized latents ready for the decoder.

---

ltx23_distilled_sigmas *Official distilled sigma schedule*

Description

The distilled LTX sigma values (with terminal zero appended), as published in the Apache-2.0 diffusers reference (pipelines/ltx2/utils.py).

Usage

```
ltx23_distilled_sigmas()
```

Value

Numeric vector of length 9.

---

ltx23_downsample1d *Anti-aliasing 1D downsampler (low-pass then stride)*

Description

Anti-aliasing 1D downsampler (low-pass then stride)

Usage

```
ltx23_downsample1d(ratio = 2L, kernel_size = NULL)
```

Arguments

ratio	Integer. Downsampling ratio.
kernel_size	Integer or NULL (default 6*ratio rounded even).

Value

Module whose forward(x) returns x low-pass filtered and decimated by ratio along the time axis.

---

ltx23_encode_audio *Encode audio into normalized, packed conditioning latents*

Description

Pads or trims the waveform so the latent length equals audio_num_frames (mel frames 4L - 3, mirroring the decoder's target_frames), computes the log-mel, encodes in argmax mode, packs, and normalizes with the checkpoint statistics.

Usage

```
ltx23_encode_audio(audio_vae, wav, audio_num_frames, frontend = NULL)
```

Arguments

audio_vae	An ltx23_audio_vae (with encoder weights).
wav	Matrix [2, samples] in [-1, 1] at 16 kHz (see ltx23_read_audio).
audio_num_frames	Integer. Target latent length.
frontend	Optional prebuilt ltx23_audio_mel_frontend.

Value

Packed normalized latents [1, audio_num_frames, 128] (float32).

---

ltx23_encode_video_frames *Encode pixel frames to normalized video latents*

Description

VAE encode in "argmax" mode (the distribution mean), then normalize with the checkpoint's per-channel statistics — the exact inverse of the decode path.

Usage

```
ltx23_encode_video_frames(vae, frames)
```

Arguments

vae	An ltx23_video_vae.
frames	Tensor [1, 3, F, H, W] in [-1, 1] (see ltx23_preprocess_frames).

Value

Normalized latents [1, 128, F', H/32, W/32] (float32).

`ltx23_feed_forward` *LTX feed-forward layer*

Description

Linear -> GELU (tanh approximation) -> Linear with 4x hidden dim, matching diffusers `FeedForward(activation_fn="ge`
state-dict names (net.0.proj, net.2).

Usage

```
ltx23_feed_forward(dim, mult = 4L)
```

Arguments

`dim`	Integer. Input/output dimension.
`mult`	Integer. Hidden dimension multiplier.

Value

Module whose forward(x) returns the projected states passed through a tanh-approximated GELU, a tensor of the same shape as `x`.

`ltx23_fp8_linear` *FP8 linear layer*

Description

Weight lives as float8_e4m3fn plus a float32 scale in plain module fields (so `$to(device)` moves only the bias); the forward pass ships 1 byte/param to the input's device, upcasts, rescales, and runs `nnf_linear`.

Usage

```
ltx23_fp8_linear(out_features, in_features, bias = TRUE)
```

Arguments

`bias`	Logical.
`out_features, in_features`	Integers.

Value

Module whose forward(x) returns the linear projection of x, with the fp8 weight bytes transferred and cast up to the compute dtype for the matmul. Same result as an nn_linear of the same shape, at a quarter of the resident weight bytes.

---

ltx23_get_timestep_embedding *Sinusoidal timestep embedding*

Description

DDPM-style sinusoidal embedding. LTX uses flip_sin_to_cos=TRUE (cos first) and downscale_freq_shift=0.

Usage

```
ltx23_get_timestep_embedding(timesteps, embedding_dim, flip_sin_to_cos = TRUE,
                             downscale_freq_shift = 0, max_period = 10000)
```

Arguments

timesteps	1D tensor of timestep values.
embedding_dim	Integer. Output embedding size.
flip_sin_to_cos	Logical. Put cos before sin.
downscale_freq_shift	Numeric. Frequency delta control.
max_period	Numeric. Maximum embedding frequency period.

Value

Tensor [N, embedding_dim].

---

ltx23_is_fp8_cast_key *Test whether a mapped DiT key is in the official fp8 cast set*

Description

Test whether a mapped DiT key is in the official fp8 cast set

Usage

```
ltx23_is_fp8_cast_key(mapped_key)
```

Arguments

mapped_key	Character vector of mapped (diffusers-style) parameter names.

Value

Logical vector.

---

ltx23_kaiser_sinc_filter1d

Kaiser sinc low-pass filter kernel

Description

Kaiser sinc low-pass filter kernel

Usage

```
ltx23_kaiser_sinc_filter1d(cutoff, half_width, kernel_size)
```

Arguments

cutoff	Numeric. Normalized cutoff in (0, 0.5].
half_width	Numeric. Transition band half width.
kernel_size	Integer.

Value

Tensor [kernel_size].

---

ltx23_latent_upsampler

LTX-2.3 latent upsampler model

Description

Latents [B, 128, F, H, W] -> [B, 128, F, 2H, 2W].

Usage

```
ltx23_latent_upsampler(in_channels = 128L, mid_channels = 1024L,
                       num_blocks_per_stage = 4L)
```

Arguments

in_channels	Integer. Latent channels.
mid_channels	Integer.
num_blocks_per_stage	Integer.

Value

Module whose forward(hidden_states) returns the 2x spatially upscaled latent, a tensor with the same batch, channel and frame counts and doubled height and width.

---

ltx23_load_group *Stream a checkpoint key group into a module*

Description

Reads tensors one at a time from an open checkpoint and copies them into the matching parameters/buffers of `module`. Destination names are derived by `map_key`; `$copy_()` handles any dtype/device conversion, so the module may already live on its target device in its target dtype.

Usage

```
ltx23_load_group(ckpt, keys, module, map_key = identity, verbose = TRUE,
                 gc_every = 50L)
```

Arguments

ckpt	An `ltx23_checkpoint`.
keys	Character vector of checkpoint keys to load (one group from `ltx23_split_keys`).
module	A torch nn_module to populate.
map_key	Function mapping a checkpoint key to the module's parameter/buffer name, or NA to skip the key deliberately.
verbose	Logical. Report progress and coverage.
gc_every	Integer. Run `gc()` after this many tensors.

Value

Invisibly, a list with `unmapped` (checkpoint keys that found no destination), `skipped` (keys the mapper declined), and `unfilled` (module parameters/buffers never written). A perfectly loaded group has zero `unmapped` and zero `unfilled`.

`ltx23_load_pipeline` *Load the LTX-2.3 generation components from a single-file checkpoint*

Description

Builds the transformer, connectors, video VAE, audio VAE, and vocoder with the LTX 2.3 configuration and streams the checkpoint weights into them. The Gemma3 text encoder ships separately (see `load_gemma3_text_encoder`).

Usage

```
ltx23_load_pipeline(checkpoint_path, device = "cuda", dtype = "bfloat16",
                    transformer_device = "cpu",
                components = c("dit", "connectors", "vae", "audio_vae", "vocoder"),
                    pin = TRUE, attn_chunk = NULL, phase_offload = TRUE,
                    verbose = TRUE)
```

Arguments

`checkpoint_path`	Path to the single-file checkpoint (e.g. `ltx-2.3-22b-distilled-1.1.safetensors`) or to an fp8 artifact directory produced by `ltx23_quantize_fp8`. With the fp8 artifact, the transformer loads with CPU-resident fp8 weights that stream to `device` during the forward pass.
`device`	Character. Device for the small components (VAEs, vocoder, connectors) and, with fp8, the transformer residents.
`dtype`	Character. "bfloat16" (checkpoint native) or "float32".
`transformer_device`	Character. Device for the transformer weights when loading the plain (non-fp8) checkpoint.
`components`	Character vector. Which components to load.
`pin`	Logical. Pin fp8 host memory (fp8 artifact only).
`attn_chunk`	Integer or NULL. Query-chunk size for attention (see `ltx23_set_attn_chunk`).
`phase_offload`	Logical. Load the small components (connectors, VAEs, vocoder) to the CPU; the pipeline moves each onto the compute device only for its phase.
`verbose`	Logical.

Value

A list with the loaded modules and the checkpoint config, class `ltx23_pipeline`.

ltx23_load_transformer_fp8

Load the LTX-2.3 transformer with FP8 weights

Description

Builds the transformer, swaps the official cast-set linears for `ltx23_fp8_linear`, loads fp8 weights CPU-side (optionally pinned) and everything else as bfloat16 on `device`. Sets `options(diffuseR.block_gc = TRUE)` so the transformer runs per-block garbage collection over the dequantized temporaries.

Usage

```
ltx23_load_transformer_fp8(ckpt, device = "cuda", pin = TRUE, verbose = TRUE,
                           ...)
```

Arguments

`ckpt`	An fp8 `ltx23_checkpoint` (`ltx23_open_fp8_checkpoint`).
`device`	Character. Device for the resident (non-fp8) weights.
`pin`	Logical. Pin the fp8 host memory for faster transfers.
`verbose`	Logical.
`...`	Passed to `ltx23_transformer` (tiny test configs).

Value

The loaded `ltx23_transformer`.

ltx23_load_transformer_nf4

Load the LTX-2.3 transformer with resident NF4 weights

Description

Builds the transformer, swaps the cast-set linears for `ltx23_nf4_linear`, and loads everything onto `device`: at ~4.5 bits/parameter the whole 22B transformer stays GPU-resident, avoiding per-step weight transfers.

Usage

```
ltx23_load_transformer_nf4(ckpt, device = "cuda", verbose = TRUE, ...)
```

Arguments

ckpt	An NF4 ltx23_checkpoint (ltx23_open_fp8_checkpoint reads any shard artifact).
device	Character.
verbose	Logical.
...	Passed to ltx23_transformer (tiny test configs).

Value

The loaded ltx23_transformer.

---

ltx23_load_upsampler *Load the LTX-2.3 spatial upscaler weights*

Description

The checkpoint keys match this module tree directly.

Usage

```
ltx23_load_upsampler(path, device = "cuda", dtype = "bfloat16", verbose = TRUE)
```

Arguments

path	Path to e.g. ltx-2.3-spatial-upscaler-x2-1.1.safetensors.
verbose	Logical.
device, dtype	Placement for the loaded model.

Value

The loaded ltx23_latent_upsampler.

---

ltx23_map_audio_vae_key *Map an official audio VAE checkpoint key to the R module name*

Description

Map an official audio VAE checkpoint key to the R module name

Usage

```
ltx23_map_audio_vae_key(key)
```

Arguments

key	Character. Checkpoint key.

Value

Character.

---

ltx23_map_connector_key *Map an official connectors checkpoint key to the R module name*

Description

Map an official connectors checkpoint key to the R module name

Usage

```
ltx23_map_connector_key(key)
```

Arguments

key	Character. Checkpoint key.

Value

Character. Module parameter name.

---

ltx23_map_dit_key *Map an official DiT checkpoint key to the R module name*

Description

Applies the official-to-diffusers renames for the LTX-2.3 transformer (cf. diffusers scripts/convert_ltx2_to_diffusers.py). Our module tree matches the diffusers names, so this is the full mapping.

Usage

```
ltx23_map_dit_key(key)
```

Arguments

key	Character. Checkpoint key (with or without the `model.diffusion_model.` prefix).

Value

Character. Module parameter/buffer name.

ltx23_map_vae_key *Map an official VAE checkpoint key to the R module name*

Description

The official checkpoint stores the encoder/decoder as flat block lists (down_blocks.0-8 / up_blocks.0-8) where downsamplers/upsamplers and the mid block are separate entries; diffusers (and this port) nest them. Index mapping per diffusers convert_ltx2_to_diffusers.py.

Usage

```
ltx23_map_vae_key(key)
```

Arguments

key	Character. Checkpoint key (with or without "vae." prefix).

Value

Character. Module parameter/buffer name.

ltx23_map_vocoder_key *Map an official vocoder checkpoint key to the R module name*

Description

Map an official vocoder checkpoint key to the R module name

Usage

```
ltx23_map_vocoder_key(key)
```

Arguments

key	Character. Checkpoint key.

Value

Character. Module parameter/buffer name.

ltx23_mel_stft *Causal log-mel spectrogram with checkpoint-loaded bases*

Description

Causal log-mel spectrogram with checkpoint-loaded bases

Usage

```
ltx23_mel_stft(filter_length = 512L, hop_length = 80L, window_length = 512L,
               num_mel_channels = 64L)
```

Arguments

`filter_length`, `hop_length`, `window_length`, `num_mel_channels`
: Integers.

Value

Module whose forward(waveform) returns the log-mel spectrogram [B, n_mels, frames], clamped at 1e-5 before the log.

ltx23_memory_profile *Get an LTX-2.3 memory profile*

Description

Selects transformer precision, component placement, and attention chunking for the available VRAM. Measured on an RTX 5060 Ti (16 GB): fp8 streaming peaks ~11.6 GB (without phase offloading) at 512x320x49; NF4 keeps the whole 22B transformer resident (~12.5 GB) and removes the ~21 GB/step PCIe weight streaming. The NF4 profile renders 1280x704x121 with audio in ~23 min at a 15.7 GB peak (tiled VAE decode, in-place feed-forward GELU, and the default `diffuseR.attn_budget` of 1.5e8 all required at that size).

Usage

```
ltx23_memory_profile(vram_gb = NULL)
```

Arguments

`vram_gb`
: Numeric or NULL (auto-detect free VRAM).

Details

precision "nf4" Weights resident on the GPU; fastest steps; about 9 percent weight round-trip error.

precision "fp8" Weights CPU-resident, streamed per linear; near-bf16 quality; each step pays the PCIe transfer.

Value

Named list with device/dtype placement, `attn_chunk`, `pin_weights`, and resolution caps.

`ltx23_nf4_dequantize` *Dequantize NF4 data to a float tensor*

Description

Dequantize NF4 data to a float tensor

Usage

```
ltx23_nf4_dequantize(packed, absmax, shape, dtype = torch::torch_bfloat16(),
                     chunk_elements = 8388608L, out = NULL)
```

Arguments

`packed`	uint8 tensor of packed index pairs.
`absmax`	float32 tensor of per-block scales.
`shape`	Integer vector. Original tensor shape.
`dtype`	Target torch dtype.
`chunk_elements`	Integer. Elements dequantized per slice (bounds the int64 index temporary).
`out`	Optional preallocated tensor of `shape` to write into (avoids allocating a fresh weight tensor per call).

Value

Tensor of `shape` in `dtype` on the input's device.

ltx23_nf4_linear *NF4 linear layer*

Description

Packed weights and per-block scales are registered as buffers, so they move with the module (uint8 packs are untouched by dtype conversions). The forward pass dequantizes on the weight's device.

Usage

```
ltx23_nf4_linear(out_features, in_features, bias = TRUE)
```

Arguments

bias	Logical.
out_features, in_features	Integers.

Value

Module whose forward(x) returns the linear projection of x, dequantizing the NF4 weight into a reusable buffer first. Same result as an nn_linear of the same shape, at roughly an eighth of the resident weight bytes.

ltx23_nf4_quantize *Quantize a tensor to NF4*

Description

Quantize a tensor to NF4

Usage

```
ltx23_nf4_quantize(x)
```

Arguments

x	Float tensor (any shape; total elements must be a multiple of 128, i.e. two 64-element blocks - always true for the LTX linears).

Value

List with packed (uint8, two indices per byte) and absmax (float32, one per 64-element block).

ltx23_normalize_latents *Normalize latents with the VAE's per-channel statistics*

Description

Normalize latents with the VAE's per-channel statistics

Usage

```
ltx23_normalize_latents(latents, latents_mean, latents_std)
```

Arguments

latents	Tensor [B, C, F, H, W].
latents_mean, latents_std	Tensors [C].

Value

Normalized latents.

ltx23_open_checkpoint *Open an LTX-2.3 checkpoint*

Description

Opens a single-file LTX checkpoint lazily (header only), validates the `model_version` metadata, and parses the embedded component configuration.

Usage

```
ltx23_open_checkpoint(path, require_version = "2.3")
```

Arguments

path	Path to the checkpoint .safetensors file.
require_version	Character. Required `model_version` prefix (default "2.3"). Set to NULL to skip the check.

Value

An object of class `ltx23_checkpoint`: a list with `handle` (safetensors reader), `keys`, `version`, `config` (parsed component configs, or NULL), and `path`.

Examples

```
## Not run:
ckpt <- ltx23_open_checkpoint("ltx-2.3-22b-distilled-1.1.safetensors")
str(ltx23_split_keys(ckpt$keys), max.level = 1)

## End(Not run)
```

`ltx23_open_fp8_checkpoint` *Open an FP8 shard directory as a checkpoint*

Description

Presents the sharded fp8 artifact through the same interface as `ltx23_open_checkpoint` so the group loaders work unchanged.

Usage

```
ltx23_open_fp8_checkpoint(dir)
```

Arguments

`dir`	The fp8 artifact directory (with manifest.json).

Value

An `ltx23_checkpoint`.

`ltx23_per_channel_rms_norm` *Per-channel RMS normalization*

Description

Normalizes by the root-mean-square across the channel dimension (dim 2 of [B, C, F, H, W]); no learned parameters.

Usage

```
ltx23_per_channel_rms_norm(eps = 1e-08)
```

Arguments

`eps`	Numeric. Stability epsilon.

Value

Module whose forward(x) returns x divided by its per-channel root mean square, a tensor of the same shape.

`ltx23_per_token_rms_norm`

Per-token RMS normalization over the channel axis

Description

Per-token RMS normalization over the channel axis

Usage

```
ltx23_per_token_rms_norm(x, eps = 1e-06)
```

Arguments

x	Tensor [B, S, C, L] of stacked per-layer hidden states.
eps	Numeric. Stability epsilon.

Value

Tensor of the same shape.

`ltx23_prepare_conditioned_latents`

Build conditioned initial latents and the conditioning mask

Description

i2v (`cond_latents` has one latent frame): the encoded frame is repeated across all latent frames and only latent frame 0 is marked conditioned. Continuation (k latent frames): the prefix tokens are replaced and marked. Unconditioned positions start as pure noise.

Usage

```
ltx23_prepare_conditioned_latents(cond_latents, latent_frames, latent_height,
                                  latent_width, noise, cond_noise_scale = 0)
```

Arguments

`cond_latents`	Normalized condition latents [1, 128, k, H', W'] from `ltx23_encode_video_frames`.
`noise`	Tensor [1, 128, F', H', W'] of standard noise (caller provides so seeding stays in one place).
`cond_noise_scale`	Numeric. Optional partial noising of the conditioned tokens (diffusers `noise_scale`, default 0).
`latent_frames`, `latent_height`, `latent_width`	Integers. Full latent geometry of the generation.

Value

list(latents [1, S, 128] float32 packed, conditioning_mask [1, S] float32 packed).

`ltx23_preprocess_frames` *Preprocess an image (or frame stack) for VAE encoding*

Description

Mirrors the diffusers VideoProcessor: bilinear resize so the shorter relative side matches, center-crop to the exact target, and scale to [-1, 1].

Usage

```
ltx23_preprocess_frames(x, height, width)
```

Arguments

`x`	Path to a PNG/JPEG, or an array [H, W, 3] (values in [0, 1]), or a [F, H, W, 3] array of frames.
`height`, `width`	Integers. Target size (multiples of 32).

Value

Float32 tensor [1, 3, F, height, width] in [-1, 1].

ltx23_quantize_fp8 *Quantize an LTX-2.3 checkpoint to FP8 shards*

Description

Streams the single-file bf16 checkpoint tensor by tensor. DiT attention/FFN linear weights are stored as float8_e4m3fn with a float32 absmax/448 per-tensor scale (<key>_scale sibling); everything else is copied through unchanged. Output shards carry the original key names plus a manifest for skip-if-exists.

Usage

```
ltx23_quantize_fp8(checkpoint_path, output_dir = NULL, shard_bytes = 1.9e+09,
                   force = FALSE, verbose = TRUE)
```

Arguments

checkpoint_path	Source .safetensors (46 GB bf16 single file).
output_dir	Output directory for shards + manifest; NULL (the default) resolves under tools::R_user_dir("diffuseR", "data").
shard_bytes	Numeric. Target shard size in bytes. The default 1.9e9 keeps every shard under the 2^31-byte (~2.15 GB) ceiling that stock CRAN safetensors can read. Pass a larger value (e.g. 4e9) only for local builds you will read back with a fork-patched safetensors.
force	Logical. Re-quantize even if a valid manifest exists.
verbose	Logical.

Value

Invisibly, the manifest list.

ltx23_quantize_nf4 *Quantize an LTX-2.3 checkpoint to NF4 shards*

Description

Same streaming layout and cast policy as ltx23_quantize_fp8, but cast-set weights are stored as NF4 (<key> packed uint8 + <key>_absmax float32 blocks + the original shape recovered from the model config at load time). Non-cast tensors are copied through unchanged. The manifest carries format = "nf4".

Usage

```
ltx23_quantize_nf4(checkpoint_path, output_dir = NULL, shard_bytes = 1.9e+09,
                   force = FALSE, verbose = TRUE)
```

Arguments

checkpoint_path	Source .safetensors (bf16 single file).
output_dir	Output directory for shards + manifest; NULL (the default) resolves under tools::R_user_dir("diffuseR", "data").
shard_bytes	Numeric. Target shard size in bytes. The default 1.9e9 keeps every shard under the 2^31-byte (~2.15 GB) ceiling that stock CRAN safetensors can read. Pass a larger value (e.g. 4e9) only for local builds you will read back with a fork-patched safetensors.
force	Logical. Re-quantize even if a valid manifest exists.
verbose	Logical.

Value

Invisibly, the manifest list.

---

ltx23_read_audio *Read an audio file as 16 kHz stereo PCM*

Description

Decodes MP3/WAV/etc. via av to 16-bit PCM at the target rate and parses the RIFF container in base R.

Usage

```
ltx23_read_audio(path, sample_rate = 16000L)
```

Arguments

path	Audio file.
sample_rate	Integer.

Value

Matrix [2, samples] in [-1, 1].

ltx23_read_tail_frames

Read the trailing frames of a video file

Description

Extracts the last n frames of an MP4 (via av) for use as continuation conditioning.

Usage

```
ltx23_read_tail_frames(path, n = 9L)
```

Arguments

path	Video file.
n	Integer. Trailing frame count.

Value

Array [n, H, W, 3] in [0, 1].

ltx23_release_dequant_buffers

Release the NF4 dequantization buffers

Description

Frees the cached per-shape weight buffers (e.g. before decoding at high resolution).

Usage

```
ltx23_release_dequant_buffers()
```

Value

Invisibly, NULL.

ltx23_rms_norm *RMS normalization*

Description

Variance is computed in float32; the result is cast back to the input dtype (or the weight dtype when elementwise affine).

Usage

```
ltx23_rms_norm(dim, eps = 1e-06, elementwise_affine = TRUE)
```

Arguments

dim	Integer. Normalized dimension size.
eps	Numeric. Stability epsilon.
elementwise_affine	Logical. Learn a scale weight.

Value

Module whose forward(x) returns x RMS-normalized over the last axis and cast back to the input dtype, a tensor of the same shape.

ltx23_rotary_pos_embed *LTX-2.3 audio/video rotary position embedder*

Description

Computes RoPE cos/sin frequency tensors from spatiotemporal patch coordinates. Video coordinates are 3D (frames scaled to seconds via fps, height, width in pixel space); audio coordinates are 1D (seconds). Coordinates are patch boundaries [start, end); the midpoint is used as the position.

Usage

```
ltx23_rotary_pos_embed(dim, patch_size = 1L, patch_size_t = 1L,
                       base_num_frames = 20L, base_height = 2048L,
                       base_width = 2048L, sampling_rate = 16000L,
                       hop_length = 160L, scale_factors = c(8L, 32L, 32L),
                       theta = 10000, causal_offset = 1L, modality = "video",
                       double_precision = TRUE, rope_type = "split",
                       num_attention_heads = 32L)
```

Arguments

`dim`	Integer. Rotary dimension (attention head dim x heads for split type at model level; see reference).
`scale_factors`	Integer vector. VAE (time, height, width) scale factors.
`theta`	Numeric. RoPE theta.
`causal_offset`	Integer. Temporal offset for the causal VAE (first frame has stride 1).
`modality`	"video" or "audio".
`double_precision`	Logical. Compute base frequencies in float64.
`rope_type`	"split" (LTX 2.3) or "interleaved".
`num_attention_heads`	Integer. Needed for the split layout.
`patch_size`, `patch_size_t`	Integers. Spatial/temporal patch sizes.
`base_num_frames`, `base_height`, `base_width`	Integers. Base grid the coordinates are normalized against.
`sampling_rate`, `hop_length`	Integers. Audio spectrogram params.

Value

Module whose forward(coords, device) returns `list(cos_freqs, sin_freqs)`, the two rotary tables to apply to queries and keys.

`ltx23_rotary_pos_embed_1d` *1D rotary embeddings for the text connectors*

Description

1D rotary embeddings for the text connectors

Usage

```
ltx23_rotary_pos_embed_1d(dim, base_seq_len = 4096L, theta = 10000,
                          double_precision = TRUE, rope_type = "split",
                          num_attention_heads = 32L)
```

Arguments

dim	Integer. Rotary dimension (connector inner dim).
base_seq_len	Integer. Base sequence length for normalization.
theta	Numeric. RoPE theta.
double_precision	Logical. Compute base frequencies in float64.
rope_type	"split" (LTX-2.3) or "interleaved".
num_attention_heads	Integer. For the split per-head layout.

Value

Module whose forward(batch_size, pos, device) returns `list(cos_freqs, sin_freqs)`, the 1-D rotary tables for a sequence of length `pos`.

`ltx23_set_attn_chunk` *Set the attention query-chunk size across a transformer*

Description

R torch has no fused attention, so the [B, H, S, S] matrix materializes; chunking queries bounds the peak. NULL disables chunking.

Usage

```
ltx23_set_attn_chunk(transformer, chunk)
```

Arguments

transformer	An `ltx23_transformer`.
chunk	Integer or NULL.

Value

Invisibly, the transformer.

ltx23_snake_beta *SnakeBeta activation*

Description

`x + (1 / (beta + eps)) * sin(x * alpha)^2` with per-channel log-scale alpha/beta parameters.

Usage

```
ltx23_snake_beta(channels, eps = 1e-09)
```

Arguments

`channels`	Integer.
`eps`	Numeric.

Value

Module whose forward(hidden_states) returns the Snake activation `x + sin(alpha * x)^2 / beta`, a tensor of the same shape as the input.

ltx23_split_keys *Split checkpoint keys by component*

Description

Splits the flat key space of an LTX single-file checkpoint into its component groups. Connector tensors live under the `model.diffusion_model.` prefix alongside the transformer, plus a top-level `text_embedding_projection.` group; both are routed to the `connectors` component.

Usage

```
ltx23_split_keys(keys)
```

Arguments

`keys`	Character vector of checkpoint tensor names.

Value

Named list of character vectors: `dit`, `connectors`, `vae`, `audio_vae`, `vocoder`, and `other` (anything unrecognized; should be empty).

ltx23_stage2_distilled_sigmas *Stage-2 distilled sigma schedule (two-stage refinement)*

Description

Stage-2 distilled sigma schedule (two-stage refinement)

Usage

```
ltx23_stage2_distilled_sigmas()
```

Value

Numeric vector of length 4.

ltx23_tail_latents *Slice the trailing latent frames of a generation for chaining*

Description

Cuts the last `k` latent frames out of a result's video latents, in the [1, 128, k, H', W'] layout that `txt2vid_ltx2(condition_latents = )` consumes, so one chunk can seed the next without leaving latent space: no decode, no re-encode, no video round-trip.

Usage

```
ltx23_tail_latents(result, k = 2L, latent_shape = NULL)
```

Arguments

`result`	A `txt2vid_ltx2` result list (uses its `latents` and `latent_shape`), or the packed latents tensor [1, S, 128] itself (then `latent_shape` is required).
`k`	Integer. Trailing latent frames to keep (default 2 = the standard 9-pixel-frame conditioning prefix).
`latent_shape`	Integer vector c(frames, height, width) of the latent geometry; only needed when `result` is a raw tensor.

Details

Semantics caveat: a latent frame sliced from inside a sequence represents 8 pixel frames, while a fresh VAE encode of a k-frame tail represents 1 + 8(k - 1) pixel frames with its first latent in first-frame form. The frozen prefix the next generation sees is therefore not identical to the pixel-path prefix; compare both on real content before relying on latent-only joins.

Value

Normalized latents [1, 128, k, H', W'] (float32), ready for `txt2vid_ltx2(condition_latents = )`.

`ltx23_text_connectors` *LTX-2.3 text connectors*

Description

Takes raw stacked per-layer text encoder hidden states and produces the video and audio text embeddings for the DiT: per-token RMS norm, per-modality sqrt(dim ratio) rescaling and projection, then a per-modality 1D connector transformer.

Usage

```
ltx23_text_connectors(caption_channels = 3840L, text_proj_in_factor = 49L,
                      video_connector_num_attention_heads = 32L,
                      video_connector_attention_head_dim = 128L,
                      video_connector_num_layers = 8L,
                      video_connector_num_learnable_registers = 128L,
                      video_gated_attn = TRUE,
                      audio_connector_num_attention_heads = 32L,
                      audio_connector_attention_head_dim = 64L,
                      audio_connector_num_layers = 8L,
                      audio_connector_num_learnable_registers = 128L,
                      audio_gated_attn = TRUE,
                      connector_rope_base_seq_len = 4096L, rope_theta = 10000,
                      rope_double_precision = TRUE, rope_type = "split",
                      video_hidden_dim = 4096L, audio_hidden_dim = 2048L,
                      proj_bias = TRUE)
```

Arguments

`caption_channels`
: Integer. Text encoder hidden size (3840 for Gemma3-12B).

`text_proj_in_factor`
: Integer. Number of stacked hidden states (num_layers + 1 = 49 for Gemma3-12B).

`video_connector_num_learnable_registers`
: Integer or NULL.

`video_gated_attn`
: Logical.

`audio_connector_num_learnable_registers`
: Integer or NULL.

`audio_gated_attn`
: Logical.

`proj_bias` Logical. Projection bias (TRUE for LTX-2.3).

`video_connector_num_attention_heads`, `video_connector_attention_head_dim`, `video_connector_num_layers`
Video connector shape (LTX-2.3: 32 x 128, 8 layers).

`audio_connector_num_attention_heads`, `audio_connector_attention_head_dim`, `audio_connector_num_layers`
Audio connector shape (LTX-2.3: 32 x 64, 8 layers).

`connector_rope_base_seq_len`, `rope_theta`, `rope_double_precision`, `rope_type`
RoPE config.

`video_hidden_dim`, `audio_hidden_dim`
Integers. Projection targets (DiT inner dims: 4096 / 2048).

Value

Module whose forward(text_encoder_hidden_states, attention_mask) returns `list(video_text_embedding, audio_text_embedding, attention_mask)`: the caption states adapted for the video and audio cross-attention streams, plus the binary mask to use with them.

---

ltx23_tone_map_latents

Sigmoid tone mapping for latents

Description

Compresses the latent dynamic range (cf. diffusers `tone_map_latents`). `compression` 0 is identity, 1 is the full effect.

Usage

```
ltx23_tone_map_latents(latents, compression)
```

Arguments

`latents` Tensor.

`compression` Numeric in [0, 1].

Value

Tone-mapped latents.

`ltx23_transformer` *LTX-2.3 video transformer model*

Description

Dual-stream audio/video DiT. Text embeddings arrive already projected to the video (`inner_dim`) and audio (`audio_inner_dim`) dimensions by the connector modules.

Usage

```
ltx23_transformer(in_channels = 128L, out_channels = 128L, patch_size = 1L,
                  patch_size_t = 1L, num_attention_heads = 32L,
                  attention_head_dim = 128L, cross_attention_dim = 4096L,
                  vae_scale_factors = c(8L, 32L, 32L), pos_embed_max_pos = 20L,
                  base_height = 2048L, base_width = 2048L, gated_attn = TRUE,
                  cross_attn_mod = TRUE, audio_in_channels = 128L,
                  audio_out_channels = 128L, audio_patch_size = 1L,
                  audio_patch_size_t = 1L, audio_num_attention_heads = 32L,
                  audio_attention_head_dim = 64L,
                  audio_cross_attention_dim = 2048L, audio_scale_factor = 4L,
                  audio_pos_embed_max_pos = 20L, audio_sampling_rate = 16000L,
                  audio_hop_length = 160L, audio_gated_attn = TRUE,
                  audio_cross_attn_mod = TRUE, num_layers = 48L,
                  norm_eps = 1e-06, rope_theta = 10000,
                  rope_double_precision = TRUE, causal_offset = 1L,
                  timestep_scale_multiplier = 1000,
                  cross_attn_timestep_scale_multiplier = 1000,
                  rope_type = "split", perturbed_attn = TRUE)
```

Arguments

`cross_attention_dim`
: Integer. Video text embedding dimension.

`vae_scale_factors`
: Integer vector. VAE (time, height, width) scales.

`audio_cross_attention_dim`
: Integer. Audio text embedding dimension.

`num_layers`
: Integer. Transformer block count.

`norm_eps`
: Numeric. Norm epsilon.

`rope_type`
: "split" (LTX-2.3) or "interleaved".

`in_channels`, `out_channels`
: Integers. Video latent channels.

`patch_size`, `patch_size_t`
: Integers. Video patch sizes.

num_attention_heads, attention_head_dim
: Video attention shape.

pos_embed_max_pos, base_height, base_width
: RoPE base grid.

audio_in_channels, audio_out_channels
: Integers. Audio latent channels.

audio_patch_size, audio_patch_size_t
: Integers. Audio patch sizes.

audio_num_attention_heads, audio_attention_head_dim
: Audio attention shape.

audio_scale_factor, audio_pos_embed_max_pos, audio_sampling_rate, audio_hop_length
: Audio latent grid parameters.

rope_theta, rope_double_precision, causal_offset
: RoPE parameters.

timestep_scale_multiplier, cross_attn_timestep_scale_multiplier
: Timestep scaling (inputs arrive already scaled; the ratio modulates the a2v/v2a gates).

gated_attn, cross_attn_mod, audio_gated_attn, audio_cross_attn_mod, perturbed_attn
: LTX-2.3 feature flags (all TRUE for the 2.3 checkpoints).

Value

Module whose forward(hidden_states, ...) returns `list(sample, audio_sample)`: the predicted velocity for the video latent tokens and, when the audio branch is active, for the audio latent tokens (`audio_sample` is NULL otherwise).

---

ltx23_transformer_block

LTX-2 transformer block

Description

Dual-stream (video + audio) block: modulated self-attention per modality, text cross-attention per modality (with LTX-2.3 query and key/value modulation), bidirectional audio-video cross-attention with global+per-block modulation, and modulated feed-forward.

Usage

```
ltx23_transformer_block(dim, num_attention_heads, attention_head_dim,
                        cross_attention_dim, audio_dim,
                        audio_num_attention_heads, audio_attention_head_dim,
                        audio_cross_attention_dim, video_gated_attn = TRUE,
                        video_cross_attn_adaln = TRUE, audio_gated_attn = TRUE,
                        audio_cross_attn_adaln = TRUE, eps = 1e-06,
                        elementwise_affine = FALSE, rope_type = "split",
                        perturbed_attn = TRUE)
```

Arguments

cross_attention_dim	Integer. Text embedding dim for video.
audio_cross_attention_dim	Integer. Text embedding dim for audio.
eps	Numeric. Norm epsilon.
elementwise_affine	Logical. Block norms carry weights (FALSE for LTX).
rope_type	"split" or "interleaved".
perturbed_attn	Logical. Enable the STG perturbation arguments.
dim, audio_dim	Integers. Video/audio stream dimensions.
num_attention_heads, attention_head_dim	Video attention shape.
audio_num_attention_heads, audio_attention_head_dim	Audio attention shape.
video_gated_attn, audio_gated_attn	Logicals. Per-head output gates.
video_cross_attn_adaln, audio_cross_attn_adaln	Logicals. LTX-2.3 text cross-attention modulation (9 mod params instead of 6).

Value

Module whose forward(hidden_states, ...) returns `list(hidden_states, audio_hidden_states)`, the video and audio streams after self-attention, cross-attention and the feed-forward, each the same shape as its input.

---

ltx23_tune_gc *Tune the torch CUDA allocator for large-resident inference*

Description

Stops the allocator GC storm (cf. ~/skills/torch torch-jit-gc-performance.md): lantern proactively calls R's gc() whenever reserved memory exceeds `torch.cuda_allocator_reserved_rate` (default 0.20) of the card. With ~75\ weights that fires on nearly every allocation. Raising the rate to the actual footprint is safe here because the LTX hot loops compute into persistent scratch buffers (near-zero per-step garbage). Also raises the host-allocation GC threshold and defaults `PYTORCH_CUDA_ALLOC_CONF` to expandable segments. User-set options win.

Usage

```
ltx23_tune_gc(footprint_gb = 12, total_gb = NULL)
```

Arguments

footprint_gb	Numeric. Expected resident GPU footprint in GB (NF4 transformer: ~12).
total_gb	Numeric or NULL (auto-detect total VRAM).

Details

start_torch() reads the gate options exactly once, so setting them after torch has started is inert on its own. The three CUDA gates are therefore also pushed into the live allocator here (the .flux_gc_gates pattern), which makes this function effective whenever it runs. The host-side torch.threshold_call_gc has no live setter; the package defaults it in .onLoad so torch reads it at init in any session that loads diffuseR before running torch ops.

Value

Invisibly, the applied reserved rate (NULL if skipped).

---

ltx23_upsample1d *Anti-aliasing 1D upsampler (transposed low-pass)*

Description

Anti-aliasing 1D upsampler (transposed low-pass)

Usage

```
ltx23_upsample1d(ratio = 2L, kernel_size = NULL, window_type = "kaiser",
                 persistent = TRUE)
```

Arguments

ratio	Integer. Upsampling ratio.
kernel_size	Integer or NULL.
window_type	"kaiser" (BigVGAN default) or "hann" (final resampler).
persistent	Logical. Register the filter as a buffer (present in checkpoints); FALSE stores the computed filter as a plain field.

Value

Module whose forward(x) returns x interpolated up by ratio along the time axis, with the filter padding trimmed off.

`ltx23_video_decoder3d` *LTX-2.3 video decoder*

Description

Latents [B, 128, F, H, W] -> pixel video [B, 3, 8F - 7, 32H, 32W]. Block channel lists are given encoder-side (as in the config) and reversed internally; `upsample_type` is indexed directly.

Usage

```
ltx23_video_decoder3d(in_channels = 128L, out_channels = 3L,
                      block_out_channels = c(256L, 512L, 512L, 1024L),
                      spatio_temporal_scaling = c(TRUE, TRUE, TRUE, TRUE),
                      layers_per_block = c(4L, 6L, 4L, 2L, 2L),
                      upsample_type = NULL, patch_size = 4L, patch_size_t = 1L,
                      resnet_norm_eps = 1e-06, is_causal = FALSE,
                      upsample_residual = c(FALSE, FALSE, FALSE, FALSE),
                      upsample_factor = c(2L, 2L, 1L, 2L),
                      spatial_padding_mode = "zeros")
```

Arguments

`block_out_channels`
: Integer vector (config order).

`spatio_temporal_scaling`
: Logical vector per up block.

`layers_per_block`
: Integer vector (config order; first entry is the mid block after reversal).

`upsample_type` Character vector per up block (not reversed).

`resnet_norm_eps`
: Numeric.

`is_causal` Logical. FALSE for LTX (symmetric temporal padding).

`upsample_residual`
: Logical vector per up block.

`upsample_factor`
: Integer vector per up block.

`spatial_padding_mode`
: Character.

`in_channels`, `out_channels`
: Integers. Latent and pixel channels.

`patch_size`, `patch_size_t`
: Integers.

Value

Module whose forward(hidden_states, causal) returns the decoded pixel tensor [B, 3, F, H, W], with the final patch axes flattened back into height and width.

ltx23_video_downsampler3d

Pixel-unshuffle 3D downsampler

Description

Conv followed by space/time-to-channel rearrangement, plus a grouped channel-mean residual of the same rearrangement.

Usage

```
ltx23_video_downsampler3d(in_channels, out_channels, stride = c(1L, 1L, 1L),
                          spatial_padding_mode = "zeros")
```

Arguments

`stride`	Length-3 integer vector (t, h, w).
`spatial_padding_mode`	Character.
`in_channels, out_channels`	Integers.

Value

Module whose forward(hidden_states, causal) returns the space-to-depth downsampled tensor plus its residual: spatial and temporal extents shrink by `stride`, channels grow to match.

ltx23_video_down_block3d

LTX video down block

Description

ResNet stack (at the input channel count) followed by a pixel-unshuffle downsampler that also changes the channel count.

Usage

```
ltx23_video_down_block3d(in_channels, out_channels = NULL, num_layers = 1L,
                         resnet_eps = 1e-06, spatio_temporal_scale = TRUE,
                         downsample_type = "spatiotemporal",
                         spatial_padding_mode = "zeros")
```

Arguments

num_layers	Integer. ResNet count.
resnet_eps	Numeric.
spatio_temporal_scale	Logical. Whether to downsample at all.
downsample_type	"spatial", "temporal", or "spatiotemporal".
spatial_padding_mode	Character.
in_channels, out_channels	Integers.

Value

Module whose forward(hidden_states, causal) returns the stage output: the resnet stack applied in sequence, then the optional downsampler.

---

ltx23_video_encoder3d *LTX-2.3 video encoder*

Description

Pixel video [B, 3, F, H, W] -> latent statistics [B, 2 * latent_channels, F/8, H/32, W/32] (mean and a uniform log-var channel broadcast across the latent channels).

Usage

```
ltx23_video_encoder3d(in_channels = 3L, out_channels = 128L,
                      block_out_channels = c(256L, 512L, 1024L, 1024L),
                      spatio_temporal_scaling = c(TRUE, TRUE, TRUE, TRUE),
                      layers_per_block = c(4L, 6L, 4L, 2L, 2L),
                      downsample_type = NULL, patch_size = 4L,
                      patch_size_t = 1L, resnet_norm_eps = 1e-06,
                      is_causal = TRUE, spatial_padding_mode = "zeros")
```

Arguments

block_out_channels	Integer vector. Per-block output channels.
spatio_temporal_scaling	Logical vector per block.
layers_per_block	Integer vector (blocks then mid).
downsample_type	Character vector per block.

`resnet_norm_eps`
: Numeric.

`is_causal`
: Logical.

`spatial_padding_mode`
: Character.

`in_channels`, `out_channels`
: Integers. Pixel and latent channels.

`patch_size`, `patch_size_t`
: Integers. Pixel patchification.

Value

Module whose forward(hidden_states, causal) returns the encoded video latent [B, C, F, H, W], with the last channel repeated to carry the per-channel scale expected downstream.

---

ltx23_video_mid_block3d *LTX video mid block*

Description

A plain ResNet stack at a fixed channel count.

Usage

```
ltx23_video_mid_block3d(in_channels, num_layers = 1L, resnet_eps = 1e-06,
                        spatial_padding_mode = "zeros")
```

Arguments

`in_channels`
: Integer.

`num_layers`
: Integer.

`resnet_eps`
: Numeric.

`spatial_padding_mode`
: Character.

Value

Module whose forward(hidden_states, causal) returns the bottleneck output, a tensor of the same shape as the input.

ltx23_video_resnet_block3d

LTX 3D ResNet block

Description

PerChannelRMSNorm -> SiLU -> causal conv, twice, with a LayerNorm + 1x1 Conv3d shortcut when the channel count changes.

Usage

```
ltx23_video_resnet_block3d(in_channels, out_channels = NULL, eps = 1e-06,
                           spatial_padding_mode = "zeros")
```

Arguments

`eps`	Numeric. Shortcut LayerNorm epsilon.
`spatial_padding_mode`	Character.
`in_channels`, `out_channels`	Integers.

Value

Module whose forward(inputs, causal) returns `inputs` plus the residual branch, a tensor of the same shape.

ltx23_video_upsampler3d

Pixel-shuffle 3D upsampler

Description

Conv followed by channel-to-space/time rearrangement, with an optional channel-repeat residual and an upscale factor that divides the conv output channels.

Usage

```
ltx23_video_upsampler3d(in_channels, stride = c(1L, 1L, 1L), residual = FALSE,
                        upscale_factor = 1L, spatial_padding_mode = "zeros")
```

Arguments

in_channels	Integer.
stride	Length-3 integer vector (t, h, w).
residual	Logical. Add the rearranged input as a residual.
upscale_factor	Integer.
spatial_padding_mode	Character.

Value

Module whose forward(hidden_states, causal) returns the depth-to-space upsampled tensor: spatial and temporal extents grow by stride, channels shrink to match.

---

ltx23_video_up_block3d

LTX video up block

Description

Optional channel-changing conv-in ResNet, pixel-shuffle upsampler, then a ResNet stack at the output channel count.

Usage

```
ltx23_video_up_block3d(in_channels, out_channels = NULL, num_layers = 1L,
                       resnet_eps = 1e-06, spatio_temporal_scale = TRUE,
                       upsample_type = "spatiotemporal",
                       upsample_residual = FALSE, upscale_factor = 1L,
                       spatial_padding_mode = "zeros")
```

Arguments

num_layers	Integer.
resnet_eps	Numeric.
spatio_temporal_scale	Logical.
upsample_type	"spatial", "temporal", or "spatiotemporal".
upsample_residual	Logical.
upscale_factor	Integer.
spatial_padding_mode	Character.
in_channels, out_channels	Integers.

Value

Module whose forward(hidden_states, causal) returns the stage output: the optional input projection and upsampler, then the resnet stack applied in sequence.

`ltx23_video_vae` *LTX-2.3 video VAE*

Description

Encoder + decoder + per-channel latent statistics (loaded from the checkpoint's `per_channel_statistics`). The checkpoint's `scaling_factor` is 1.0, so latent (de)normalization is purely the per-channel affine map.

Usage

```
ltx23_video_vae(in_channels = 3L, out_channels = 3L, latent_channels = 128L,
                block_out_channels = c(256L, 512L, 1024L, 1024L),
                decoder_block_out_channels = c(256L, 512L, 512L, 1024L),
                layers_per_block = c(4L, 6L, 4L, 2L, 2L),
                decoder_layers_per_block = c(4L, 6L, 4L, 2L, 2L),
                spatio_temporal_scaling = c(TRUE, TRUE, TRUE, TRUE),
                decoder_spatio_temporal_scaling = c(TRUE, TRUE, TRUE, TRUE),
                downsample_type = NULL, upsample_type = NULL,
                upsample_residual = c(FALSE, FALSE, FALSE, FALSE),
                upsample_factor = c(2L, 2L, 1L, 2L), patch_size = 4L,
                patch_size_t = 1L, resnet_norm_eps = 1e-06,
                encoder_causal = TRUE, decoder_causal = FALSE,
                encoder_spatial_padding_mode = "zeros",
                decoder_spatial_padding_mode = "zeros")
```

Arguments

`latent_channels`
: Integer.

`resnet_norm_eps`
: Numeric.

`in_channels, out_channels`
: Integers. Pixel channels.

`block_out_channels, layers_per_block, spatio_temporal_scaling, downsample_type`
: Encoder configuration (see `ltx23_video_encoder3d`).

`decoder_block_out_channels, decoder_layers_per_block, decoder_spatio_temporal_scaling, upsample_type, upsample_residual, upsample_factor`
: Decoder configuration (see `ltx23_video_decoder3d`).

patch_size, patch_size_t
: Integers. Pixel patchification.

encoder_causal, decoder_causal
: Logicals. Temporal padding modes.

encoder_spatial_padding_mode, decoder_spatial_padding_mode
: Characters.

Value

Module bundling the video encoder and decoder. Its forward(z) is `decode(z)`, returning pixels for a latent; `$encode()` and `$decode()` are callable separately.

`ltx23_vocoder` *LTX-2.3 vocoder stage*

Description

Mel spectrogram [B, C, T, M] -> waveform [B, out_channels, samples]. Channel and mel dims are flattened into conv channels; each upsample stage halves the channel count and averages three parallel ResNet branches.

Usage

```
ltx23_vocoder(in_channels = 128L, hidden_channels = 1536L, out_channels = 2L,
              upsample_kernel_sizes = c(11L, 4L, 4L, 4L, 4L, 4L),
              upsample_factors = c(5L, 2L, 2L, 2L, 2L, 2L),
              resnet_kernel_sizes = c(3L, 7L, 11L),
            resnet_dilations = list(c(1L, 3L, 5L), c(1L, 3L, 5L), c(1L, 3L, 5L)),
              antialias_ratio = 2L, antialias_kernel_size = 12L,
              final_bias = FALSE)
```

Arguments

in_channels
: Integer. Flattened input channels (C * mel bins / 1).

hidden_channels
: Integer.

out_channels
: Integer.

resnet_kernel_sizes
: Integer vector.

resnet_dilations
: List of integer vectors.

final_bias
: Logical.

upsample_kernel_sizes, upsample_factors
: Integer vectors.

antialias_ratio, antialias_kernel_size
: Integers.

Value

Module whose forward(hidden_states, time_last) returns the synthesized waveform [B, 1, samples] for a mel spectrogram.

`ltx23_vocoder_resblock` *Vocoder ResNet block (AMP)*

Description

Dilated conv pairs, each preceded by an anti-aliased SnakeBeta activation, with residual connections.

Usage

```
ltx23_vocoder_resblock(channels, kernel_size = 3L, dilations = c(1L, 3L, 5L),
                       antialias_ratio = 2L, antialias_kernel_size = 12L)
```

Arguments

`channels`	Integer.
`kernel_size`	Integer.
`dilations`	Integer vector.
`antialias_ratio`, `antialias_kernel_size`	Integers.

Value

Module whose forward(x) returns `x` after the dilated convolution pairs have been added back as residuals, a tensor of the same shape.

`ltx23_vocoder_with_bwe` *LTX-2.3 vocoder with bandwidth extension*

Description

Full mel [B, 2, T, 64] -> 48 kHz stereo waveform pipeline: 16 kHz vocoder, causal mel re-analysis, BWE vocoder residual added to a Hann-resampled skip path, clamped to [-1, 1].

Usage

```
ltx23_vocoder_with_bwe(in_channels = 128L, hidden_channels = 1536L,
                       out_channels = 2L,
                       upsample_kernel_sizes = c(11L, 4L, 4L, 4L, 4L, 4L),
                       upsample_factors = c(5L, 2L, 2L, 2L, 2L, 2L),
                       resnet_kernel_sizes = c(3L, 7L, 11L),
                       resnet_dilations = NULL, bwe_in_channels = 128L,
                       bwe_hidden_channels = 512L,
                       bwe_upsample_kernel_sizes = c(12L, 11L, 4L, 4L, 4L),
                       bwe_upsample_factors = c(6L, 5L, 2L, 2L, 2L),
                       bwe_resnet_kernel_sizes = c(3L, 7L, 11L),
                       bwe_resnet_dilations = NULL, filter_length = 512L,
                       hop_length = 80L, window_length = 512L,
                       num_mel_channels = 64L, input_sampling_rate = 16000L,
                       output_sampling_rate = 48000L)
```

Arguments

`out_channels` Integer. Audio channels.

`hop_length` Integer. Mel analysis hop.

`in_channels`, `bwe_in_channels`
: Integers. Flattened mel input channels.

`hidden_channels`, `bwe_hidden_channels`
: Integers.

`upsample_kernel_sizes`, `upsample_factors`, `bwe_upsample_kernel_sizes`, `bwe_upsample_factors`
: Integer vectors. Per-stage transposed-conv configs.

`resnet_kernel_sizes`, `bwe_resnet_kernel_sizes`
: Integer vectors.

`resnet_dilations`, `bwe_resnet_dilations`
: Lists of integer vectors.

`filter_length`, `window_length`, `num_mel_channels`
: Integers. Mel re-analysis configuration.

`input_sampling_rate`, `output_sampling_rate`
: Integers.

Value

Module whose forward(mel_spec) returns the bandwidth-extended waveform [B, 1, samples], trimmed to the sample count implied by the input frames and the rate ratio.

`memory_flux` *FLUX Memory Profiles*

Description

VRAM-based execution profiles for the FLUX.1-schnell pipeline. The 12B transformer runs NF4 (~7 GB) or fp8 (~12 GB), phase-onloaded to the GPU for denoise; the T5-XXL text encoder phase-onloads to the GPU (bfloat16, pinned) on 14 GB+ cards and computes on the CPU (float32) below that, where its ~9.8 GB encode phase does not fit.

`memory_ltx23` *LTX-2.3 Memory Profiles and CUDA GC Tuning*

Description

Memory management for running the 22B LTX-2.3 transformer on limited VRAM, built on the patterns proven in the whisper and chatterbox packages: torch allocator GC tuning before the first CUDA op, fp8 CPU-resident streaming weights, query-chunked attention, and phase-sequential component placement.

`models2devices` *models2devices*

Description

This function sets up the model directory, device configuration, and data types for diffusion models. It checks the validity of the model name and devices, detects model type, and downloads the model if necessary.

Usage

```
models2devices(model_name, devices = "cpu", unet_dtype_str = NULL,
               download_models = FALSE)
```

Arguments

`model_name`	A character string representing the name of the model to be used.
`devices`	A character string or a named list specifying the devices for different components of the model.
`unet_dtype_str`	A character string specifying the data type for the UNet model.
`download_models`	Logical indicating whether to download models if they are not found.

Value

A list containing the device configuration, UNet data type, and CPU/CUDA devices.

nf4_ltx23 *NF4 Weight Storage for the LTX-2.3 Transformer*

Description

4-bit NormalFloat quantization (the QLoRA scheme: per-block absmax normalization against a 16-level quantile code, two indices packed per byte). At ~4.5 bits/parameter the 22B transformer fits in about 12.5 GB, small enough to stay resident on a 16 GB GPU: no per-step PCIe weight streaming, at a small quality cost relative to fp8. Quantization and dequantization are pure torch ops (bucketize, index_select) - no custom kernels.

offload_to_cpu *Offload Module to CPU*

Description

Moves a torch module and all its parameters to CPU.

Usage

```
offload_to_cpu(module, gc = TRUE)
```

Arguments

module	A torch nn_module.
gc	Logical. Run garbage collection after offload.

Value

The module (modified in place).

Examples

```
if (torch::torch_is_installed()) {
  model <- torch::nn_linear(4, 2)
  offload_to_cpu(model)
}
```

post_quant_conv *Post Quant Conv*

Description

This function applies a quantized convolution operation to an input tensor. It is typically used in the context of image post processing, particularly in generative models like Stable Diffusion XL.

Usage

```
post_quant_conv(x, dtype, device)
```

Arguments

x	Input tensor to be processed.
dtype	Data type for the tensor (e.g., "torch_float16" or "torch_float32").
device	Device on which the tensor is located (e.g., "cpu" or "cuda").

Value

Processed tensor after applying the quantized convolution.

preprocess_image *Preprocess image for Stable Diffusion*

Description

Preprocess image for Stable Diffusion

Usage

```
preprocess_image(input, device = "cpu", width = 512, height = 512)
```

Arguments

input	File path to .jpg or .png, or a 3D array
device	Target device for torch ("cpu" or "cuda")
width	Desired width of the output image
height	Desired height of the output image

Value

Torch tensor of shape c(1, 3, 512, 512), scaled to c(-1, 1)

print.bpe_tokenizer *Print BPE Tokenizer*

Description

Print BPE Tokenizer

Usage

```
## S3 method for class 'bpe_tokenizer'
print(x, ...)
```

Arguments

x	A bpe_tokenizer object.
...	Additional arguments (ignored).

Value

Invisibly returns x. Called for the side effect of printing a summary of the tokenizer to the console.

print.diffuseR_resident *Print a resident handle*

Description

Print a resident handle

Usage

```
## S3 method for class 'diffuseR_resident'
print(x, ...)
```

Arguments

x	A diffuseR_resident handle.
...	Ignored.

Value

Invisibly x. Called for the side effect of printing a one-block summary to the console.

quantize_flux *FLUX Transformer Quantization and Loading*

Description

Quantize the 12B FLUX transformer to NF4 (~7 GB, GPU-resident on 16 GB cards) or fp8 (~12 GB, CPU-resident and streamed per forward), and load any format back into `flux_transformer`. Reuses the LTX-2.3 quantization machinery (`ltx23_nf4_quantize`, `ltx23_nf4_linear`, `ltx23_fp8_linear`); only the cast set and the diffusers directory layout are FLUX-specific.

quant_conv *Quant Conv*

Description

This function applies a quantized convolution operation to an input tensor. It is typically used in the context of image processing, particularly in generative models like Stable Diffusion.

Usage

```
quant_conv(x, dtype, device)
```

Arguments

`x`	Input tensor to be processed.
`dtype`	Data type for the tensor (e.g., "torch_float16" or "torch_float32").
`device`	Device on which the tensor is located (e.g., "cpu" or "cuda").

Value

Processed tensor after applying the quantized convolution.

qwen3_encoder *Qwen3 encoder stack*

Description

Defaults are the Qwen3-4B configuration used by FLUX.2 klein. The module tree mirrors the checkpoint keys (`model.embed_tokens`, `model.layers.*`, `model.norm`); the tied LM head carries no weights of its own and is not implemented.

Usage

```
qwen3_encoder(vocab_size = 151936L, hidden_size = 2560L,
              intermediate_size = 9728L, num_hidden_layers = 36L,
              num_attention_heads = 32L, num_key_value_heads = 8L,
              head_dim = 128L, rope_theta = 1e+06, rms_norm_eps = 1e-06)
```

Arguments

`rope_theta`	Numeric.
`rms_norm_eps`	Numeric.
`vocab_size`, `hidden_size`, `intermediate_size`, `num_hidden_layers`	Integers.
`num_attention_heads`, `num_key_value_heads`, `head_dim`	Integers.

Value

Module whose forward(input_ids, attention_mask = NULL, out_layers) returns a list of hidden-state tensors [B, S, hidden], one per requested layer (a value of k means the state after k layers, matching HF `output.hidden_states[k]`). Runs only to `max(out_layers)`. `input_ids` are 1-based.

qwen3_text_encoder *Qwen3 Text Encoder*

Description

Fresh R port of the Qwen3 decoder stack from HuggingFace transformers (Apache-2.0, src/transformers/models/qwen3/), used by FLUX.2 klein as its text encoder (Qwen3-4B: 36 layers, hidden 2560, 32 query / 8 KV heads, head_dim 128, SwiGLU 9728, RoPE theta 1e6). The pipeline consumes mid-stack hidden states (layers 9, 18, 27 for klein-4B) concatenated per token, so the forward runs only as deep as the last requested layer; the LM head is never needed (embeddings are tied). Causal attention with the tokenizer's padding mask, matching the reference exactly.

qwen_bpe_tokenizer *Load a Qwen2 byte-level BPE tokenizer*

Description

Load a Qwen2 byte-level BPE tokenizer

Usage

```
qwen_bpe_tokenizer(tokenizer_path)
```

Arguments

tokenizer_path	Path to a tokenizer.json (or a directory containing one).

Value

A qwen_tokenizer object.

recommend *Recommend a precision and device configuration for a model*

Description

One VRAM-and-capability-aware recommendation for every diffuseR model. The policy:

Usage

```
recommend(model = c("sd21", "sdxl", "flux1", "flux2", "zimage", "ltx"),
          vram_gb = NULL, st_caps = NULL, host_ram_gb = NULL)
```

Arguments

model	"sd21", "sdxl", "flux1", "flux2", "zimage", or "ltx".
vram_gb	Numeric or NULL. Free VRAM in GB; auto-detected via nvidia-smi when NULL.
st_caps	NULL or a named logical list with bfloat16 and/or float8_e4m3fn - the safetensors READ capabilities. NULL probes the installed safetensors.
host_ram_gb	Numeric or NULL. Available host RAM in GB; auto-detected (Linux MemAvailable) when NULL, NA where undetectable.

Details

- nf4 is the default tier for the quantized families (flux1, flux2, zimage, ltx). Its weights are packed uint8 plus float32 blocks in sub-2 GB shards, which every safetensors reads, so it always loads. The SD models ship no quantized weights; their floor is fp16 with placement varying by VRAM.
- When the card has room for a higher-quality tier (fp8 or bf16) AND the installed safetensors can *read* that dtype (`.st_can_read`), that tier is recommended instead.
- When the card has room but safetensors cannot read the tier, nf4 is recommended and the fork suggestion is surfaced in `note` (never an error).

This is the policy engine; it does no disk I/O and does not know which artifacts are built. Loaders reconcile the recommendation with what is on disk (see `flux_load_pipeline`). Thresholds are validated on an RTX 5060 Ti (16 GB) and are deliberately conservative elsewhere. Video sizing for `"ltx"` is coarse here; the LTX pipeline uses `ltx23_memory_profile` for frame-aware placement.

The pinning decision: phase-swapped weights are page-locked host copies (see `staging`) that transfer at DMA rate - but pinned pages are unswappable, so on small-RAM machines they turn memory pressure into OOM kills. `pin` is TRUE when available host RAM covers the model's pinned set twice over, FALSE below that, FALSE on the cpu tier (nothing stages), and TRUE when RAM cannot be detected (page-locking already fails soft per component). The LTX pipeline, the Gemma3 encoder, and the FLUX-family image loaders (flux1, flux2, zimage) take `pin` arguments and stage pinned weights (see `staging`); the SD-family loaders place components statically and do not phase-swap, so `pin` is inert for them. `options(diffuseR.pin_staging)` is the global switch.

Value

A list with `model`, `precision`, `devices` (named component -> device map), `offload` (phase-offloading logical), `max_pixels`, `text_device`, `attn_chunk`, `vram_gb`, `pin` (page-lock the phase-swapped host copies), `pinned_set_gb` (estimated pinned bytes), `host_ram_gb`, `fork_suggested` (logical), and `note` (the fork suggestion string, or NULL).

Examples

```
# Stating vram_gb and st_caps makes the policy deterministic: no GPU
# and no installed safetensors needed.
r <- recommend("flux1", vram_gb = 16,
               st_caps = list(bfloat16 = TRUE, float8_e4m3fn = FALSE))
r$precision       # "nf4": fp8 fits the card, but cannot be read
r$fork_suggested  # TRUE
cat(r$note)       # the fork-or-nf4 message

# Same card, once safetensors can read float8
recommend("flux1", vram_gb = 16,
          st_caps = list(bfloat16 = TRUE, float8_e4m3fn = TRUE))$precision

# Auto-detect VRAM and probe the installed safetensors
str(recommend("flux2"))
```

reshard_safetensors *Re-shard a large safetensors file into sub-2 GB shards*

Description

Splits a single `.safetensors` file into diffusers-style shards (`<base>-00001-of-000NN.safetensors` plus a `<base>.safetensors.index.json` weight map) so it loads on stock CRAN safetensors, which overflows a 32-bit offset on any file at or above 2^31 bytes. Reading the oversize source requires a fork-patched safetensors (a build-machine step); the shards it writes are fork-free to read. Used to host large fp16 diffusers weights (e.g. the 5 GB SDXL UNet) unchanged, without quantization.

Usage

```
reshard_safetensors(input, output_dir, base = "diffusion_pytorch_model",
                    shard_bytes = 1.9e+09, verbose = TRUE)
```

Arguments

`input`	Path to the source `.safetensors` file, or a directory containing `<base>.safetensors`.
`output_dir`	Output directory for the shards + index.
`base`	Shard basename (default `"diffusion_pytorch_model"`).
`shard_bytes`	Target shard size; the default 1.9e9 keeps each shard under the ~2.15 GB ceiling.
`verbose`	Logical.

Value

Invisibly, the path to the written index.json.

resident_activate *Bring a resident pipeline onto the GPU*

Description

Copies every pinned component to the handle's bound device by DMA and verifies the result tensor-by-tensor. A failure rolls back to the pinned host state; a rollback that cannot itself be verified leaves the handle broken.

Usage

```
resident_activate(res)
```

Arguments

`res`	A `diffuseR_resident` handle.

Details

What activation does depends on how the pipeline was loaded:

- `phase_offload = TRUE` (the default, and what the `txt2img_*` functions expect): no bulk transfer. The render moves each component on as its phase begins and back off as it ends, from these same pinned copies, so pre-loading them would be undone within one phase. Activation is the ownership claim.
- `phase_offload = FALSE`: every component is copied to the card up front and stays there across renders. This is the fast path, and it is checked against free VRAM first.

The distinction is not cosmetic. FLUX.1's pinned set is 15.73 GB, which does not fit a 15.47 GiB card – bulk-onloading it OOMs even though the phased render fits comfortably. So `state` is a claim about who owns the card, not a measurement of what is on it; read `components_on_gpu` from `resident_status` for the measurement.

Value

Invisibly the handle, with state "active".

`resident_deactivate` *Release a resident pipeline's VRAM*

Description

Re-points every component at its pinned host copy and drops the GPU storage. Weights are immutable during inference, so the pinned copies are still current and this moves no bytes: it is a pointer swap plus a cache release. The handle stays loaded and can be reactivated without touching the disk.

Usage

```
resident_deactivate(res, release = TRUE)
```

Arguments

`res`	A `diffuseR_resident` handle.
`release`	Empty the CUDA caching allocator afterwards. Leave TRUE unless another handle on the same device is about to reuse the pool.

Value

Invisibly the handle, with state "inactive".

resident_generate *Generate from an active resident pipeline*

Description

Dispatches to the family's generator with the resident pipeline supplied, so no weights are re-read. The handle must be active.

Usage

```
resident_generate(res, prompt, ...)
```

Arguments

res	A diffuseR_resident handle.
prompt	Character. The text prompt.
...	Passed to txt2img_flux, txt2img_flux2, txt2img_zimage or txt2vid_ltx2.

Value

Whatever the family generator returns: an image array for the image families, a video array for ltx.

resident_load *Load a diffusion pipeline as a resident handle*

Description

Loads a pipeline once and keeps its weights page-locked on the host for the life of the handle. The GPU representation is created by resident_activate and destroyed by resident_deactivate, so a 16 GB card can hand itself between models without either one re-reading its weights from disk.

Usage

```
resident_load(model = c("flux2", "flux1", "zimage", "ltx"), device = "cuda",
              ..., verbose = TRUE)
```

Arguments

model	One of "flux1", "flux2", "zimage", "ltx".
device	Target CUDA device, e.g. "cuda" or "cuda:1".
...	Passed to the family loader (flux_load_pipeline, flux2_load_pipeline, zimage_load_pipeline or ltx23_load_pipeline). ltx requires checkpoint_path.
verbose	Print progress messages.

Details

The handle is bound to one explicit GPU at load: a bare `"cuda"` resolves to the current device now, and every later transition uses that index, so the handle cannot drift to whichever GPU happens to be current at transition time.

One caveat on multi-GPU hosts: the family loader itself runs on the *current* device, and only the residency handle is bound to `device`. Loading with `device = "cuda:1"` from a session whose current device is 0 therefore stages through GPU 0 before the first activation lands on GPU 1. Wrap the call in `torch::with_device(device = "cuda:1", ...)` when that matters.

The pipeline is left *inactive* (weights pinned on the host, no VRAM held). Call `resident_activate` before generating.

Value

A `diffuseR_resident` handle (an environment). Inspect it with `resident_status`; the fields of interest are the state, the bound device, the component names, and the pinned host byte count.

See Also

`resident_activate`, `resident_status`

Examples

```
## Not run:
res <- resident_load("flux2")
resident_activate(res)
img <- resident_generate(res, "a cat in a spacesuit", seed = 7)
resident_deactivate(res) # VRAM freed, weights stay pinned in RAM
resident_activate(res) # fast: DMA copy, no disk
resident_unload(res)

## End(Not run)
```

`resident_status` *Status of a resident handle*

Description

Status of a resident handle

Usage

```
resident_status(res)
```

Arguments

`res`	A `diffuseR_resident` handle.

Value

A list with `model`, `state`, `device`, `components` (character vector of pinned component names), `pinned_bytes` (page-locked host bytes held), `gpu_allocated` and `gpu_reserved` (bytes the CUDA caching allocator reports live and held for this process, NA without CUDA), `components_on_gpu` (how many components are *actually* resident right now), `loaded_at`, and `last_error` (NULL unless a transition failed).

`state` is the handle's claim on the card; `components_on_gpu` is the measurement. They disagree by design after a render on a `phase_offload = TRUE` pipeline, which returns each component to pinned host memory as its phase finishes: the handle stays "active" (it still owns the card's budget and can render again without touching disk) while `components_on_gpu` is 0. Schedule on the measurement.

---

resident_unload *Drop a resident handle entirely*

Description

Releases the GPU copy if any, drops the pipeline and the pinned host storage, and marks the handle unloaded. Terminal: nothing but `resident_status` works afterwards.

Usage

```
resident_unload(res)
```

Arguments

`res`	A `diffuseR_resident` handle.

Value

Invisibly the handle, with state "unloaded".

---

rope_flux *FLUX Rotary Positional Embeddings*

Description

Fresh R port of the FLUX rotary positional embedding scheme from the diffusers reference implementation (Apache-2.0, src/diffusers/models/transformers/transformer_flux.py FluxPosEmbed and src/diffusers/models/embeddings.py get_1d_rotary_pos_embed / apply_rotary_emb). FLUX uses the interleaved adjacent-pair convention (use_real_unbind_dim = -1) with per-axis frequencies computed in float64 and applied in float32. Text tokens carry all-zero ids, so they receive the identity rotation.

rope_flux2 *FLUX.2 Position Ids and Empirical Shift*

Description

Fresh R port of the FLUX.2 position-id builders and the empirical timestep-shift formula from the diffusers reference (Apache-2.0, src/diffusers/pipelines/flux2/pipeline_flux2_klein.py). FLUX.2 uses 4-axis rotary position ids (T, H, W, L): text tokens carry only the L axis (sequence position), image latents carry H and W, and the T axis distinguishes reference images (unused for txt2img). Frequencies come from `flux_pos_embed` with `axes_dim = c(32, 32, 32, 32)` and `theta = 2000`.

rope_ltx23 *LTX-2.3 Rotary Positional Embeddings*

Description

Fresh R port of the LTX rotary positional embedding scheme from the diffusers reference implementation (Apache-2.0, src/diffusers/models/transformers/transformer_ltx2.py). LTX 2.3 uses the "split" RoPE layout everywhere; "interleaved" is kept for completeness. Frequencies are computed in float64 per the checkpoint config (frequencies_precision) and applied in float32.

rope_zimage *Z-Image Rotary Positional Embeddings and Patchify Helpers*

Description

Fresh R port of the Z-Image position scheme from the diffusers reference (Apache-2.0, src/diffusers/models/transformers/tran
RopeEmbedder, create_coordinate_grid, _patchify_image, _pad_with_ids, unpatchify). Z-Image uses 3-axis interleaved RoPE with theta 256; frequencies are built in float64 but the angles are cast to float32 before cos/sin (torch.polar on a .float() tensor), which differs measurably from the FLUX convention at large positions. Every sub-sequence is padded to a multiple of 32 (SEQ_MULTI_OF); caption positions are a 1-based ramp on axis 1 built over the padded length, image positions sit on axes 2/3 with axis 1 offset just past the caption.

save_frames *Save Video Frames as Individual Images*

Description

Save Video Frames as Individual Images

Usage

```
save_frames(video, dir, prefix = "frame_", format = "png", verbose = TRUE)
```

Arguments

video	Array of video frames [T, H, W, C].
dir	Directory to save frames in.
prefix	Character. Filename prefix (default "frame_").
format	Character. Image format: "png" or "jpg".
verbose	Logical.

Value

Invisibly returns vector of saved file paths.

save_image *Save and Display an Image from a Torch Tensor*

Description

Converts a Torch tensor to a normalized RGB image array, saves it as a PNG file, and optionally displays it in the RStudio Viewer pane using ‘grid::grid.raster()‘.

Usage

```
save_image(img, save_to = "output.png", normalize = TRUE)
```

Arguments

img	A numeric with shape ‘[3, H, W]‘.
save_to	File path for the PNG image (default is ‘"output.png"‘).
normalize	Logical; whether to normalize pixel values to ‘[0, 1]‘. Default is ‘TRUE‘.

Value

Invisibly returns the saved file path.

Examples

```
img <- array(runif(32 * 32 * 3), dim = c(32, 32, 3))
out <- file.path(tempdir(), "sample.png")
save_image(img, out)
file.exists(out)
unlink(out)
```

---

save_video *Save Video to File*

Description

Saves a video array to a file in various formats.

Usage

```
save_video(video, file, fps = 24, format = NULL, backend = "auto",
           quality = 85, verbose = TRUE)
```

Arguments

video	Array of video frames with shape [T, H, W, C] where C is 3 (RGB). Values should be in [0, 1] range.
file	Character. Output file path. Extension determines format.
fps	Numeric. Frames per second (default 24).
format	Character. Output format: "mp4", "gif", "webm", or "frames". If NULL, inferred from file extension.
backend	Character. Backend to use: "ffmpeg", "av", or "auto".
quality	Integer. Quality level 1-100 (for lossy formats).
verbose	Logical. Print progress messages.

Value

Invisibly returns the output file path.

Examples

```
video <- array(runif(4 * 16 * 16 * 3), dim = c(4, 16, 16, 3))

# Individual PNG frames need no external encoder.
frame_dir <- file.path(tempdir(), "frames")
save_video(video, frame_dir, format = "frames", verbose = FALSE)
length(list.files(frame_dir, pattern = "[.]png$"))
unlink(frame_dir, recursive = TRUE)

# MP4 and GIF need an ffmpeg binary or the 'av' package. Not shown as
# a live example: both backends hand off to an ffmpeg process that
```

```
# inherits this session's stdin, and R CMD check feeds the example
# script to R on stdin, so the encoder eats a byte of the script and
# every later example parses one character short.
# save_video(video, "output.mp4", fps = 24)
# save_video(video, "output.gif", fps = 10)
```

---

save_video_ltx23 *Save an LTX video (optionally with audio) to MP4*

Description

Uses the av package (Suggests) to encode frames and mux the audio track.

Usage

```
save_video_ltx23(video, filename, fps = 24, audio = NULL, sample_rate = 48000L,
                 verbose = TRUE)
```

Arguments

video	Array [frames, height, width, 3] in [0, 1].
filename	Output path (.mp4).
fps	Numeric.
audio	Optional numeric matrix [channels, samples] in [-1, 1].
sample_rate	Integer.
verbose	Logical.

Value

Invisibly, the filename.

---

scheduler_add_noise *Add noise to latents using DDIM scheduler*

Description

This function adds noise to the original latents according to the DDIM scheduler's diffusion process. It computes the noisy latents based on the original latents, noise, and the current timestep.

Usage

```
scheduler_add_noise(original_latents, noise, timestep, scheduler_obj)
```

Arguments

original_latents	A torch tensor representing the original latents.
noise	A torch tensor representing the noise to be added.
timestep	An integer representing the current timestep in the diffusion process.
scheduler_obj	A list containing the DDIM scheduler parameters, including alphas_cumprod and timesteps. The alphas_cumprod represents how much of the original signal remains at each timestep of the diffusion process.

Details

The noise is added according to the standard diffusion forward process formula: noised_latents = sqrt(alpha_cumprod) * original_latents + sqrt(1-alpha_cumprod) * noise

Where alpha_cumprod is the cumulative product of (1-beta) values up to the specified timestep, with beta being the noise schedule.

Value

A torch tensor containing the noised latents, which represents the original latents with the appropriate amount of noise added for the given timestep.

Examples

```
if (torch::torch_is_installed()) {
  scheduler <- ddim_scheduler_create(num_inference_steps = 5)
  latents <- torch::torch_randn(c(1, 4, 8, 8))
  noised_latents <- scheduler_add_noise(
    original_latents = latents,
    noise = torch::torch_randn_like(latents),
    timestep = scheduler$timesteps[1],
    scheduler_obj = scheduler
  )
  noised_latents$shape
}
```

---

sdxl_memory_profile *Get SDXL Memory Profile*

Description

Determines optimal memory configuration for SDXL image generation based on available VRAM.

Usage

```
sdxl_memory_profile(vram_gb = NULL)
```

Arguments

vram_gb	Numeric. Available VRAM in GB, or NULL for auto-detection.

Details

Memory profiles for SDXL:

full_gpu 16GB+ - All components on CUDA

balanced 10-12GB - UNet + decoder on CUDA, text encoders on CPU

unet_gpu 6-10GB - Only UNet on CUDA, everything else CPU

cpu_only <6GB - All on CPU

Each profile also specifies: - cfg_mode: "batched" or "sequential" (sequential halves peak memory) - cleanup: "none", "phase", or "step" (when to clear VRAM) - dtype: "float16" or "float32" - max_resolution: maximum image dimension

Value

A list with memory profile settings.

Examples

```
# A stated VRAM budget is deterministic and needs no GPU.
str(sdxl_memory_profile(vram_gb = 8))

str(sdxl_memory_profile(vram_gb = 24))

# Auto-detect free VRAM on this machine.
str(sdxl_memory_profile())
```

---

sdxl_pipeline_from_safetensors

Assemble a native SDXL pipeline from a diffusers safetensors directory

---

Description

Builds the two native CLIP text encoders, the native SDXL UNet, and the VAE decode module from a diffusers directory using the `*_from_safetensors` constructors, places each on its component device, and returns the list the `txt2img_sdxl` denoise loop expects.

Usage

```
sdxl_pipeline_from_safetensors(diffusers_dir, devices = NULL,
                               unet_dtype = NULL, verbose = TRUE)
```

Arguments

diffusers_dir	Directory with unet/, vae/, text_encoder/, text_encoder_2/ subdirectories.
devices	Named list of component devices (unet, decoder, text_encoder, text_encoder2); defaults to all-CPU. text_encoder2 defaults to the text_encoder device when unset.
unet_dtype	A torch dtype for the UNet (default float16 on CUDA, float32 on CPU).
verbose	Logical.

Details

Both encoders return their penultimate hidden state (SDXL feeds the UNet the concatenated [text_encoder (768) | text_encoder_2 (1280)] = 2048-dim penultimate embeds); text_encoder uses quick_gelu (OpenAI CLIP ViT-L) and text_encoder_2 uses exact GELU (OpenCLIP bigG). The pooled text_embeds come from text_encoder_2's full stack. The VAE decodes in float32 (the SDXL fp16 VAE overflows in fp16) and its scaling_factor is read from vae/config.json.

Value

A list with unet, decoder, text_encoder, text_encoder2, vae_scaling, and native_decode (TRUE; the decoder already applies post_quant_conv).

---

sdxl_pipeline_safetensors

Native SDXL pipeline from diffusers safetensors

Description

Assemble and run the native SDXL pipeline directly from a HuggingFace diffusers directory (unet/, vae/, text_encoder/, text_encoder_2/), with no TorchScript .pt step - so it works on Blackwell and loads the same weights everyone else uses. The counterpart to sd_pipeline_from_safetensors, adding the second text encoder (OpenCLIP ViT-bigG) and the added-conditioning embeddings (pooled text_embeds + time_ids) SDXL needs.

---

sd_pipeline_from_safetensors

Assemble a native SD pipeline from a diffusers safetensors directory

---

Description

Builds the native UNet, VAE decoder, and CLIP text encoder from a diffusers directory using the *_from_safetensors constructors, places each on its component device, and returns the $unet / $decoder / $text_encoder list the txt2img_* denoise loop expects.

Usage

```
sd_pipeline_from_safetensors(diffusers_dir, model_name = "sd21",
                             devices = NULL, unet_dtype = NULL, verbose = TRUE)
```

Arguments

diffusers_dir	Directory with unet/, vae/, text_encoder/ subdirectories.
model_name	Currently "sd21" (SDXL pending its second encoder).
devices	Named list of component devices (unet, decoder, text_encoder); defaults to all-CPU.
unet_dtype	A torch dtype for the UNet (default float16 on CUDA, float32 on CPU).
verbose	Logical.

Value

A list with unet, decoder, text_encoder.

---

sd_pipeline_safetensors *Native Stable Diffusion pipelines from diffusers safetensors*

Description

Assemble and run the native SD pipeline directly from a HuggingFace diffusers directory (unet/, vae/, text_encoder/), with no TorchScript .pt step - so it works on Blackwell and loads the same weights everyone else uses. SD21 is wired end to end here; SDXL still needs its second text encoder and added-conditioning embeddings (tracked in tasks/todo.md).

---

serve *Serve diffuseR over HTTP*

Description

Starts a blocking HTTP server that loads one model and answers OpenAI-style generation requests. Never downloads weights: if the model's artifacts are missing, startup stops with the loader's pointer to the explicit download_*() function.

Usage

```
serve(port = 7812L, model = c("flux2", "zimage", "flux1", "ltx"),
      device = "cuda", token = NULL, max_pixels = 1024L^2, max_frames = 161L,
      max_steps = 50L, max_pixel_frames = NULL, max_prompts = 32L,
      timeout = 300L, max_body = 1024L^2, warmup = TRUE)
```

Arguments

port	Integer. TCP port. Default 7812 (cornball serve range: whisper 7809, chatterbox 7810, qwen3 TTS 7811).
model	One of "flux2", "zimage", "flux1" (images) or "ltx" (video). SD 2.1/SDXL are not served yet.
device	Character. "cuda" or "cpu".
token	Character or NULL. Shared secret; when set, requests must send `Authorization: Bearer <token>`.
max_pixels	Integer. Maximum width x height accepted (images and video frames). Default 1024^2.
max_frames	Integer. Maximum video frame count. Default 161.
max_steps	Integer. Maximum image inference steps. Default 50.
max_pixel_frames	Numeric. Joint video budget: width x height x frames must stay under it (NULL = max_pixels x 121, so full- resolution clips top out at 121 frames and longer clips must shrink spatially).
max_prompts	Integer. Bound on the per-prompt connector-embed cache for "ltx" (~9 MB per entry, LRU-evicted). Default 32.
timeout	Integer. Per-connection I/O timeout in seconds.
max_body	Integer. Maximum request body bytes. Default 1 MB (bodies are JSON).
warmup	Logical. Image models: run one small generation at startup so the first request doesn't pay tracing and allocator growth. Ignored for "ltx".

Details

Endpoints:

- `GET /health` - liveness probe, returns `{"status":"ok","model":...}`. The server is single-threaded, so health only answers between requests.
- `POST /v1/images/generations` - image models. JSON body `{prompt, size, seed, steps}` (`size` like `"1024x1024"`; `n > 1` is not supported). Returns `{created, data: [{b64_json}]}` with a base64 PNG.
- `POST /v1/videos/generations` - `model = "ltx"` only. JSON body `{prompt, width, height, num_frames, frame_rate, seed}`. Returns raw `video/mp4` bytes. Video generation takes minutes: give your client a matching timeout.

The server is single-threaded and runs until interrupted. Run it under a process supervisor (systemd, tmux); an example unit ships with the package: `system.file("diffuser.service", package = "diffuseR")`.

Security: base R's `serverSocket` binds all interfaces, so the server is reachable by anything that can reach the machine. Keep it behind a firewall or reverse proxy, and/or set `token`: when set, every request must carry `Authorization: Bearer <token>` or it is refused with 401. Generation size is capped by `max_pixels`/`max_frames`; oversized requests get 400. A CUDA out-of-memory during a request answers 500 and then exits the process (status 70) so a supervisor restarts it with clean GPU state rather than serving on with stranded components.

Value

Does not return normally; runs until interrupted.

`setup_dtype` *Set up dtype based on device configuration*

Description

Set up dtype based on device configuration

Usage

```
setup_dtype(devices, unet_dtype_str)
```

Arguments

`devices`	A character string or a named list specifying the devices for model components.
`unet_dtype_str`	A character string specifying the data type for the UNet model.

Value

A torch dtype object based on the main computation device.

`staging` *Pinned Staging for Phase-Sequential Components*

Description

Phase offloading moves each large component (transformer, connectors, VAEs, vocoder, text encoders) between CPU and GPU every render. From pageable memory those copies run through the driver's bounce buffer at a fraction of PCIe speed; page-locked (pinned) host memory transfers by DMA at full rate. Each component's parameters and buffers are pinned once at load; onload swaps every tensor to a non-blocking GPU copy of its pinned source, and offload simply re-points the tensors at the still-valid pinned copies — weights are immutable during inference, so offload moves no bytes at all.

Details

Costs: the model's host copies become non-swappable for the life of the pipeline (no extra RAM - set_data repoints the same tensors), and page-locking adds ~9s to pipeline load. Measured post byte-LUT (768x512x49, NF4, RTX 5060 Ti): ~7s saved per render (warm renders 64-66s pageable vs 57-59s pinned; denoise and decode identical, the delta is pure transfer), so pinning breaks even on the second render and costs a single-render session ~2s net. On by default; page-locking failure falls back silently per component, and `options(diffuseR.pin_staging = FALSE)` before the loader opts out (e.g. under host memory pressure, where unswappable pages turn thrashing into OOM). The LTX pipeline, the Gemma3 encoder, and the FLUX-family image loaders (flux1, flux2, zimage) all stage pinned weights; `recommend` computes the RAM-aware `pin` default per model.

standardize_devices *Standardize devices configuration*

Description

This function standardizes the device configuration for model components. It checks if the devices parameter is a single string or a named list, and fills in missing components with reasonable defaults.

Usage

```
standardize_devices(devices, required_components)
```

Arguments

devices	A character string or a named list specifying the devices for model components.
required_components	A character vector of required components for the model.

Value

A named list of devices for each required component.

st_caps *safetensors read-capability probes and fork messaging*

Description

The CRAN build of safetensors 0.2.1 reads bfloat16 but cannot write it, and has no float8 support at all. Both fixes merged upstream on 2026-07-31 (mlverse/safetensors#11 for bfloat16 write, #13 for float8) without a version bump, so the installed version number cannot tell you which build you have. That is why every gate here is a runtime probe: write a tiny tensor, read it back, cache the answer. Two capabilities matter and they differ:

Details

- *write* (flux_quantize's internal .st_can_write, in quantize_flux.R): needed to BUILD a quantized artifact in that dtype.
- *read* (.st_can_read, here): needed to LOAD a hosted artifact in that dtype. This is the capability that gates user-facing recommendations. It is strictly weaker than write: CRAN safetensors reads bfloat16 it cannot write, so the write probe is the wrong signal for whether a hosted bf16 artifact will load.

Both are capability-probed, never version-pinned, so the fork requirement self-heals the day the fixes reach CRAN.

t5_encoder *T5 encoder stack*

Description

Defaults are the T5-v1.1-XXL configuration used by FLUX.

Usage

```
t5_encoder(vocab_size = 32128L, d_model = 4096L, d_kv = 64L, num_heads = 64L,
           d_ff = 10240L, num_layers = 24L,
           relative_attention_num_buckets = 32L,
           relative_attention_max_distance = 128L, layer_norm_epsilon = 1e-06)
```

Arguments

`layer_norm_epsilon`
: Numeric.

`vocab_size`, `d_model`, `d_kv`, `num_heads`, `d_ff`, `num_layers`
: Integers.

`relative_attention_num_buckets`, `relative_attention_max_distance`
: Integers. Relative position bias shape.

Value

Module whose forward(input_ids) (1-based ids [B, S]) returns the last hidden state [B, S, d_model].

t5_text_encoder *T5 Text Encoder (T5-v1.1)*

Description

Fresh R port of the T5 encoder stack from HuggingFace transformers (Apache-2.0, src/transformers/models/t5/modeling_t5.p as used by FLUX's second text encoder (T5-v1.1-XXL: 24 layers, d_model 4096, 64 heads x d_kv 64, gated-GELU FFN). Distinctives faithfully carried over: RMS layer norms (no mean subtraction), no biases anywhere, no 1/sqrt(d) attention scaling (folded into the weights), and a shared relative position bias computed once from block 1's embedding and added to every layer's attention logits. Module field names mirror the checkpoint keys (minus the `encoder.` prefix).

Details

FLUX passes no attention mask - padding tokens attend and are attended to - so none is implemented.

text_encoder2_native *Native CLIP Text Encoder 2 (OpenCLIP ViT-bigG for SDXL)*

Description

Native R torch implementation of OpenCLIP text encoder used in SDXL. Returns both hidden states and pooled output.

Usage

```
text_encoder2_native(vocab_size = 49408, context_length = 77, embed_dim = 1280,
                     num_layers = 32, num_heads = 20, mlp_dim = 5120,
                     return_penultimate = FALSE)
```

Arguments

vocab_size	Vocabulary size (default 49408)
context_length	Maximum sequence length (default 77)
embed_dim	Embedding dimension (default 1280)
num_layers	Number of transformer layers (default 32)
num_heads	Number of attention heads (default 20)
mlp_dim	MLP hidden dimension (default 5120)
return_penultimate	Return the penultimate hidden state alongside the pooled output (SDXL's hidden_states[-2] prompt embeds; the pooled output still comes from the full stack)

Value

An nn_module representing the text encoder

text_encoder2_native_from_safetensors *Build a native SDXL text encoder 2 from a diffusers safetensors directory*

Description

Reads the CLIPTextConfig from <dir>/config.json, constructs text_encoder2_native to match, and loads model.safetensors - the safetensors counterpart to the TorchScript text_encoder_2 path (no TorchScript, Blackwell-safe).

Usage

```
text_encoder2_native_from_safetensors(path, return_penultimate = TRUE,
                                      verbose = TRUE, ...)
```

Arguments

path	diffusers text_encoder_2 directory (config.json + model.safetensors) or the config.json path.
return_penultimate	Return the penultimate hidden state for the SDXL cross-attention embeds (default TRUE); the pooled output is always computed from the full stack.
verbose	Print how many parameters were loaded.
...	Overrides for text_encoder2_native args.

Value

The native text encoder 2 in eval mode.

---

text_encoder_native *Native CLIP Text Encoder*

Description

Native R torch implementation of CLIP text encoder. Replaces TorchScript for better GPU compatibility.

Usage

```
text_encoder_native(vocab_size = 49408, context_length = 77, embed_dim = 768,
                    num_layers = 12, num_heads = 12, mlp_dim = 3072,
                    apply_final_ln = TRUE, return_penultimate = FALSE,
                    gelu_type = "tanh")
```

Arguments

vocab_size	Vocabulary size (default 49408)
context_length	Maximum sequence length (default 77)
embed_dim	Embedding dimension
num_layers	Number of transformer layers
num_heads	Number of attention heads
mlp_dim	MLP hidden dimension
apply_final_ln	Whether to apply final layer norm (default TRUE). Set to FALSE to match TorchScript exports that don't include final LN.
return_penultimate	Return the second-to-last transformer block's output (hidden_states[-2], pre-final-LN) instead of the last layer. This is what SDXL feeds the UNet cross-attention; final LN is never applied to the penultimate output (default FALSE).
gelu_type	GELU variant: "tanh" (matches the TorchScript exports), "quick" (HF CLIP ViT-L, used by SDXL text_encoder and FLUX), or "exact"

Value

An nn_module representing the text encoder

`text_encoder_native_from_safetensors` *Build a native CLIP text encoder from a diffusers safetensors directory*

Description

Reads the CLIPTextConfig from `<dir>/config.json`, constructs `text_encoder_native` to match, and loads `model.safetensors` - the safetensors counterpart to the TorchScript text-encoder path (no TorchScript, Blackwell-safe). Handles SD21's OpenCLIP ViT-H and SDXL's CLIP ViT-L (which is the same checkpoint as FLUX's `text_encoder`). `apply_final_ln` governs only the forward output; the `final_layer_norm` weights load either way. Use `TRUE` for SD21 and pooled CLIP outputs, `FALSE` for the SDXL penultimate-layer prompt embeds.

Usage

```
text_encoder_native_from_safetensors(path, apply_final_ln = TRUE,
                                     verbose = TRUE, ...)
```

Arguments

`path`	diffusers text_encoder directory (config.json + model.safetensors) or the config.json path.
`apply_final_ln`	Apply the final layer norm in forward (default TRUE).
`verbose`	Print how many parameters were loaded.
`...`	Overrides for `text_encoder_native` args (e.g. `gelu_type`).

Value

The native text encoder in eval mode.

`tokenizer_qwen` *Qwen2 Byte-Level BPE Tokenizer*

Description

Pure R implementation of the Qwen2 tokenizer (HuggingFace tokenizer.json, BPE model with ByteLevel pre-tokenization), as used by FLUX.2 klein's Qwen3 text encoder. Text is split with the GPT-4-style regex, each pre-token's UTF-8 bytes are mapped through the GPT-2 byte-to-unicode table, and rank-based BPE merges produce the ids. Added tokens (`<|im_start|>`, `<think>`, ...) are split out literally before byte-level encoding.

Details

Limitation: the NFC normalizer is not applied (base R has no NFC); input is assumed to already be NFC, which holds for ordinary text.

---

tokenizer_unigram *SentencePiece Unigram Tokenizer*

Description

Pure R implementation of HuggingFace tokenizer.json files with a Unigram model (SentencePiece), as used by T5 - FLUX's second text encoder. Segmentation is Viterbi best-path over the vocab log probabilities (Kudo 2018, arXiv:1804.10959). The normalizer and Metaspace pre-tokenizer settings are read from the file.

Details

Limitation: the Precompiled charsmap normalizer (NFKC-style unicode mapping) is approximated by control-whitespace substitution only; ASCII and common latin text tokenizes identically to the reference, exotic unicode may differ.

---

tokenize_gemma3 *Tokenize text for Gemma3*

Description

Tokenize text for Gemma3

Usage

```
tokenize_gemma3(tokenizer, text, max_length = 1024L, padding = "max_length",
                return_tensors = "pt")
```

Arguments

tokenizer	Gemma3 tokenizer object.
text	Character vector of prompts.
max_length	Integer. Maximum sequence length.
padding	Character. Padding strategy ("left", "right", "max_length", "none").
return_tensors	Character. Return type ("list" or "pt" for torch tensors).

Value

List with input_ids and attention_mask.

txt2img *Generate an image from a text prompt using a diffusion pipeline*

Description

Generate an image from a text prompt using a diffusion pipeline

Usage

```
txt2img(prompt, model_name = c("sd21", "sdxl", "flux1", "flux2", "zimage"), ...)
```

Arguments

prompt	A character string prompt describing the image to generate.
model_name	Name of the model to use (e.g., '"sd21"').
...	Additional parameters passed to the diffusion process.

Value

A tensor or image object, depending on implementation.

Examples

```
## Not run:
img <- txt2img("a cat wearing sunglasses in space", device = "cuda")

## End(Not run)
```

txt2img_flux *Generate an image with FLUX.1-schnell*

Description

4-step distilled text-to-image generation (no classifier-free guidance): T5 + CLIP prompt encoding, flow-matching Euler denoising over the packed latent sequence, and 16-channel VAE decode. With phase offloading each component is the sole GPU tenant for its phase.

Usage

```
txt2img_flux(prompt, pipeline = NULL, width = 1024L, height = 1024L,
             num_inference_steps = 4L, max_sequence_length = 256L, seed = NULL,
             prompt_embeds = NULL, pooled_prompt_embeds = NULL,
             save_file = TRUE, filename = NULL, verbose = TRUE, ...)
```

Arguments

prompt	Character. The prompt.
pipeline	A flux_pipeline from flux_load_pipeline; NULL loads one (passing ... through).
num_inference_steps	Integer. Denoising steps (schnell: 4).
max_sequence_length	Integer. T5 token length (schnell: 256).
seed	Integer or NULL. Initial latents are drawn on the CPU, so a seed matches a Python diffusers run with a CPU generator.
save_file	Logical. Write a PNG.
filename	Output path (default derived from the prompt).
verbose	Logical, or one of "silent", "progress", "steps". TRUE = "steps" (full per-phase chatter), FALSE = "silent". "progress" prints a one-line generation summary plus a denoise progress bar (interactive) or periodic step ticks (captured logs).
...	Passed to flux_load_pipeline when pipeline is NULL.
width, height	Integers, divisible by 16.
prompt_embeds, pooled_prompt_embeds	Optional precomputed text embeddings (skip the text encoders).

Value

Invisibly, list(image, metadata) where image is an [H, W, 3] array in [0, 1].

---

txt2img_flux2 *Generate an image with FLUX.2 klein*

Description

Step-distilled text-to-image (klein-4B: 4 steps, no guidance): Qwen3 prompt encoding (chat template, mid-stack hidden states), FlowMatch denoising with the empirical dynamic shift, and 32-channel VAE decode through the BatchNorm latent statistics.

Usage

```
txt2img_flux2(prompt, pipeline = NULL, width = 1024L, height = 1024L,
              num_inference_steps = 4L, max_sequence_length = 512L,
              seed = NULL, prompt_embeds = NULL, save_file = TRUE,
              filename = NULL, verbose = TRUE, ...)
```

Arguments

prompt	Character. The prompt.
pipeline	A flux2_pipeline from flux2_load_pipeline; NULL loads one (passing ... through).
num_inference_steps	Integer. Denoising steps (klein-4B: 4).
max_sequence_length	Integer. Qwen3 token length (512).
seed	Integer or NULL. Latents are drawn on the CPU in the packed shape, so a seed matches a Python diffusers run with a CPU generator.
prompt_embeds	Optional precomputed [B, S, 7680] embeddings.
save_file	Logical. Write a PNG.
filename	Output path (default derived from the prompt).
verbose	Logical, or one of "silent", "progress", "steps". TRUE = "steps" (full per-phase chatter), FALSE = "silent". "progress" prints a one-line generation summary plus a denoise progress bar (interactive) or periodic step ticks (captured logs).
...	Passed to flux2_load_pipeline when pipeline is NULL.
width, height	Integers, divisible by 16.

Value

Invisibly, list(image, metadata) where image is an [H, W, 3] array in [0, 1].

---

txt2img_sd21 *Generate an image from a text prompt using a diffusion pipeline*

Description

This function generates an image based on a text prompt using the Stable Diffusion model. It allows for various configurations such as model name, device, scheduler, and more.

Usage

```
txt2img_sd21(prompt, negative_prompt = NULL, img_dim = 768, pipeline = NULL,
             devices = "auto", unet_dtype_str = NULL, download_models = FALSE,
             scheduler = "ddim", timesteps = NULL, initial_latents = NULL,
             num_inference_steps = 50, guidance_scale = 7.5, seed = NULL,
             save_file = TRUE, filename = NULL, metadata_path = NULL,
             use_native_decoder = FALSE, use_native_text_encoder = FALSE,
             use_native_unet = FALSE, diffusers_dir = NULL, ...)
```

Arguments

prompt	A character string prompt describing the image to generate.
negative_prompt	Optional negative prompt to guide the generation.
img_dim	Dimension of the output image (e.g., 512 for 512x512).
pipeline	Optional A pre-loaded diffusion pipeline. If 'NULL', it will be loaded based on the model name and devices.
devices	A named list of devices for each model component (e.g., 'list(unet = "cuda", decoder = "cpu", text_encoder = "cpu")').
unet_dtype_str	Optional A character for dtype of the unet component (typically "float16" for cuda and "float32" for cpu; float32 is available for cuda).
download_models	Logical indicating whether to download the model files if they are not found.
scheduler	Scheduler to use (e.g., '"ddim"', '"euler"').
timesteps	Optional A vector of timesteps to use.
initial_latents	Optional initial latents for the diffusion process.
num_inference_steps	Number of inference steps to run.
guidance_scale	Scale for classifier-free guidance (typically 7.5).
seed	Optional seed for reproducibility.
save_file	Logical indicating whether to save the generated image.
filename	Optional filename for saving the image. If 'NULL', a default name is generated.
metadata_path	Optional file path to save metadata.
use_native_decoder	Logical; if TRUE, uses native R torch decoder instead of TorchScript. Native decoder has better GPU compatibility (especially Blackwell).
use_native_text_encoder	Logical; if TRUE, uses native R torch text encoder instead of TorchScript. Native text encoder has better GPU compatibility (especially Blackwell).
use_native_unet	Logical; if TRUE, uses native R torch UNet instead of TorchScript. Native UNet has better GPU compatibility (especially Blackwell).
diffusers_dir	Optional path to a HuggingFace diffusers directory (with 'unet/', 'vae/', 'text_encoder/'). When set, the pipeline is built natively from safetensors (no TorchScript), via [sd_pipeline_from_safetensors()]. See [download_sd21()].
...	Additional parameters passed to the diffusion process.

Value

An image array and metadata

Examples

```
## Not run:
img <- txt2img("a cat wearing sunglasses in space", device = "cuda")

## End(Not run)
```

`txt2img_sdxl` *Generate an image from a text prompt using SDXL*

Description

Generate an image from a text prompt using SDXL

Usage

```
txt2img_sdxl(prompt, negative_prompt = NULL, img_dim = 1024, pipeline = NULL,
             devices = "auto", memory_profile = NULL, unet_dtype_str = NULL,
             download_models = FALSE, scheduler = "ddim", timesteps = NULL,
             initial_latents = NULL, num_inference_steps = 30,
             guidance_scale = 7.5, seed = NULL, save_file = TRUE,
             filename = NULL, metadata_path = NULL, use_native_decoder = FALSE,
             use_native_text_encoder = FALSE, use_native_unet = FALSE,
             diffusers_dir = NULL, verbose = TRUE, ...)
```

Arguments

`prompt`	A character string prompt describing the image to generate.
`negative_prompt`	Optional negative prompt to guide the generation.
`img_dim`	Dimension of the output image (e.g., 512 for 512x512).
`pipeline`	Optional A pre-loaded diffusion pipeline. If ‘NULL‘, it will be loaded based on the model name and devices.
`devices`	A named list of devices for each model component (e.g., ‘list(unet = "cuda", decoder = "cpu", text_encoder = "cpu")‘), or "auto" to use ‘auto_devices()‘, or NULL to use memory_profile devices.
`memory_profile`	Character or list. Memory profile for GPU-poor optimization: "auto" for auto-detection, or a profile name ("full_gpu", "balanced", "unet_gpu", "cpu_only"), or a list from ‘sdxl_memory_profile()‘. When specified, overrides devices parameter.
`unet_dtype_str`	Optional A character for dtype of the unet component (typically "float16" for cuda and "float32" for cpu; float32 is available for cuda).
`download_models`	Logical indicating whether to download the model files if they are not found.
`scheduler`	Scheduler to use (e.g., ‘"ddim"‘, ‘"euler"‘).

timesteps	Optional A vector of timesteps to use.
initial_latents	Optional initial latents for the diffusion process.
num_inference_steps	Number of inference steps to run.
guidance_scale	Scale for classifier-free guidance (typically 7.5).
seed	Optional seed for reproducibility.
save_file	Logical indicating whether to save the generated image.
filename	Optional filename for saving the image. If 'NULL', a default name is generated.
metadata_path	Optional file path to save metadata.
use_native_decoder	Logical; if TRUE, uses native R torch decoder instead of TorchScript. Native decoder has better GPU compatibility (especially Blackwell).
use_native_text_encoder	Logical; if TRUE, uses native R torch text encoder instead of TorchScript. Native text encoder has better GPU compatibility (especially Blackwell).
use_native_unet	Logical; if TRUE, uses native R torch UNet instead of TorchScript. Native UNet has better GPU compatibility (especially Blackwell).
diffusers_dir	Optional path to a diffusers safetensors directory (`unet/`, `vae/`, `text_encoder/`, `text_encoder_2/`). When supplied, the pipeline is built end-to-end from safetensors via `sdxl_pipeline_from_safetensors` (native, Blackwell-safe, no TorchScript) and the other `use_native_*` flags are ignored.
verbose	Logical. Print progress and memory status messages.
...	Additional parameters passed to the diffusion process.

Value

An image array and metadata

Examples

```
## Not run:
# Basic usage with auto-detection
img <- txt2img_sdxl("a cat wearing sunglasses in space")

# GPU-poor mode (8GB VRAM)
img <- txt2img_sdxl("a sunset over mountains", memory_profile = "unet_gpu")

# Explicit memory profile
profile <- sdxl_memory_profile(vram_gb = 8)
img <- txt2img_sdxl("a forest path", memory_profile = profile)

## End(Not run)
```

`txt2img_zimage` *Generate an image with Z-Image-Turbo*

Description

Guidance-distilled text-to-image (8 steps, no CFG): Qwen3-4B prompt encoding (thinking-enabled chat template, penultimate hidden state), FlowMatch denoising with the reversed-timestep convention, and 16-channel VAE decode. Strong at legible text rendering, English and Chinese both.

Usage

```
txt2img_zimage(prompt, pipeline = NULL, width = 1024L, height = 1024L,
               num_inference_steps = 8L, max_sequence_length = 512L,
               seed = NULL, prompt_embeds = NULL, save_file = TRUE,
               filename = NULL, verbose = TRUE, ...)
```

Arguments

`prompt`	Character. The prompt.
`pipeline`	A `zimage_pipeline` from `zimage_load_pipeline`; NULL loads one (passing `...` through).
`num_inference_steps`	Integer. Denoising steps (Turbo: 8).
`max_sequence_length`	Integer. Qwen3 token length (512).
`seed`	Integer or NULL. Latents are drawn on the CPU, so a seed matches a Python diffusers run with a CPU generator.
`prompt_embeds`	Optional precomputed [L, 2560] caption embeddings (valid tokens only).
`save_file`	Logical. Write a PNG.
`filename`	Output path (default derived from the prompt).
`verbose`	Logical, or one of "silent", "progress", "steps". TRUE = "steps" (full per-phase chatter), FALSE = "silent". "progress" prints a one-line generation summary plus a denoise progress bar (interactive) or periodic step ticks (captured logs).
`...`	Passed to `zimage_load_pipeline` when `pipeline` is NULL.
`width, height`	Integers, divisible by 16.

Value

Invisibly, `list(image, metadata)` where `image` is an [H, W, 3] array in [0, 1].

`txt2vid_ltx2` *Generate video (and audio) with LTX-2.3*

Description

Distilled text-to-video generation: encodes the prompt with Gemma3 + connectors, denoises joint audio/video latents over the official 8-step distilled schedule (no classifier-free guidance), decodes the video with the causal VAE and the audio with the audio VAE + BWE vocoder, and optionally muxes both into an MP4.

Usage

```
txt2vid_ltx2(prompt, pipeline, text_encoder = NULL, tokenizer = NULL,
             prompt_embeds = NULL, connector_embeds = NULL, width = 768L,
             height = 512L, num_frames = 121L, frame_rate = 24,
             sigmas = ltx23_distilled_sigmas(), guidance_scale = 1,
             seed = NULL, device = "cuda", dtype = "bfloat16", filename = NULL,
             max_sequence_length = 1024L, decode_video = TRUE,
             decode_audio = TRUE, two_stage = FALSE, upsampler = NULL,
             adain_factor = 1, tone_map_compression = 0, phase_offload = TRUE,
             image = NULL, condition_video = NULL, conditioning_frames = 9L,
             cond_noise_scale = 0, condition_latents = NULL,
             resident = character(), trim_frames = 0L, audio = NULL,
             verbose = TRUE)
```

Arguments

`prompt`	Character. The prompt.
`pipeline`	An `ltx23_pipeline` from `ltx23_load_pipeline`.
`prompt_embeds`	Optional precomputed list with `prompt_embeds` (raw stacked Gemma3 states) and `prompt_attention_mask`; bypasses the text encoder.
`connector_embeds`	Optional precomputed text-connector outputs: a list with `video_text_embedding`, `audio_text_embedding`, and `attention_mask` (the result of `pipeline$connectors` on the Gemma3 states). The prompt is constant across a chained track, so compute this once and pass it to every chunk: it skips the per-call connectors phase, whose GPU-side handling of the raw hidden-state stack does not fit next to a resident transformer.
`num_frames`	Integer. 8k + 1 frames (e.g. 121).
`frame_rate`	Numeric. Frames per second.
`sigmas`	Numeric vector. Denoising schedule (default: official distilled schedule; must end in 0).
`guidance_scale`	Numeric. Only 1 (no CFG) is supported; the distilled checkpoints are trained for CFG-free sampling.
`seed`	Integer or NULL.

`device` Character. Compute device for the denoising loop.

`dtype` Character. Model compute dtype ("bfloat16" or "float32").

`filename` Character or NULL. Output video path (.mp4). Audio is muxed in when the av package is available.

`max_sequence_length`
Integer. Text token length (multiple of 128).

`two_stage` Logical. Generate at half resolution, upsample the latents 2x spatially, and refine over the stage-2 schedule (resolution must then be a multiple of 64; requires `upsampler`).

`upsampler` An `ltx23_latent_upsampler` (see `ltx23_load_upsampler`).

`adain_factor` Numeric. AdaIN blend of the upsampled latents toward the stage-1 statistics (0 disables).

`tone_map_compression`
Numeric in [0, 1]. Optional latent tone mapping before stage 2.

`phase_offload` Logical. Move each small component to the compute device only for its phase (text encoding, upsampling, decoding) and back to the CPU afterwards, keeping the denoise phase as the sole GPU tenant.

`image` Optional start image for image-to-video: a PNG/JPEG path or an [H, W, 3] array in [0, 1]. The image conditions the first frame; the rest of the video is generated (reference i2v).

`condition_video`
Optional continuation source: a video path (its trailing `conditioning_frames` frames are used) or an [F, H, W, 3] array. The clip's tail becomes the frozen prefix of the new video, so the output's first `conditioning_frames` frames overlap the source (trim or crossfade when concatenating).

`conditioning_frames`
Integer. Trailing pixel frames taken from `condition_video` (8k + 1, default 9 = 2 latent frames).

`cond_noise_scale`
Numeric in [0, 1]. Optional partial noising of the conditioned tokens (0 = keep them exactly).

`condition_latents`
Optional continuation source already in latent space: normalized video latents [1, 128, k, height/32, width/32] (e.g. `ltx23_tail_latents` on a previous result), used directly as the frozen prefix with no VAE encode. Mutually exclusive with `image` and `condition_video`.

`resident` Character vector of pipeline component names ("transformer", "vae", "audio_vae", "connectors", "vocoder") to keep on the compute device after their phase instead of offloading, for callers running several generations back to back (chained chunks). Components already on the device are not re-copied on later calls.

`trim_frames` Integer. Drop this many leading pixel frames from the decoded video (and the saved file), e.g. the conditioning-head overlap of a continuation. The returned `latents` keep the full sequence (tail slicing for chaining needs it). Audio is muxed exactly as supplied, so drop any head padding from the conditioning audio when trimming.

audio — Optional conditioning audio for audio-driven generation (lip sync): a file path (decoded via av) or a matrix [2, samples] in [-1, 1] at 16 kHz. The audio is encoded into clean, frozen audio latents that the video attends to while denoising, and the original samples are muxed into the output (audio decoding is skipped).

verbose — Logical, or one of "silent", "progress", "steps". TRUE = "steps" (full per-phase chatter, per-step sigma/timing lines), FALSE = "silent". "progress" prints a one-line generation summary plus a denoise progress bar (interactive) or periodic step ticks (captured logs).

text_encoder, tokenizer — Gemma3 model and tokenizer (or paths; see load_gemma3_text_encoder and gemma3_tokenizer). Ignored when prompt_embeds is supplied.

width, height — Integers. Output resolution (multiples of 32).

decode_video, decode_audio — Logicals. Decode the respective latents (disable for latent-space work).

Value

Invisibly, a list with video (array [frames, height, width, 3] in [0, 1]), audio (matrix [2, samples] in [-1, 1]), sample_rate, the raw latents and audio_latents, and latent_shape (c(frames, height, width) of the latent geometry, for ltx23_tail_latents).

---

txt2vid_ltx23 *LTX-2.3 Text-to-Video Pipeline*

Description

Fresh R port of the LTX-2 text-to-video flow from the diffusers reference (Apache-2.0, pipelines/ltx2/pipeline_ltx2.py), specialized for the distilled LTX 2.3 checkpoints: 8-step official sigma schedule, no classifier-free guidance, joint audio-video denoising with an Euler velocity step, and audio decoding through the audio VAE and BWE vocoder to 48 kHz stereo.

---

unet_native *Native UNet for Stable Diffusion*

Description

Native R torch implementation of UNet2DConditionModel. Replaces TorchScript for better GPU compatibility.

Usage

```
unet_native(in_channels = 4L, out_channels = 4L,
            block_out_channels = c(320L, 640L, 1280L, 1280L),
            layers_per_block = 2L, cross_attention_dim = 1024L,
            attention_head_dim = 64L)
```

Arguments

`in_channels`	Input channels (default 4 for latent space)
`out_channels`	Output channels (default 4)
`block_out_channels`	Channel multipliers per block
`layers_per_block`	Number of ResBlocks per down/up block
`cross_attention_dim`	Context dimension from text encoder
`attention_head_dim`	Dimension per attention head

Value

An nn_module representing the UNet

`unet_native_from_safetensors` *Build a native SD21 UNet from a diffusers safetensors directory*

Description

The safetensors counterpart to `unet_native_from_torchscript`: constructs `unet_native` and loads its weights from `unet/diffusion_pytorch_model.safetensors` (no TorchScript, so it works on Blackwell). The default construction matches the canonical Stable Diffusion 2.1 UNet; pass constructor overrides through `...` for a variant checkpoint (the loader fails loudly on any shape mismatch, so a wrong architecture surfaces immediately rather than loading silently wrong weights).

Usage

```
unet_native_from_safetensors(path, verbose = TRUE, ...)
```

Arguments

`path`	Path to the UNet directory or its single-file checkpoint.
`verbose`	Print how many parameters were loaded.
`...`	Overrides for `unet_native` constructor args.

Value

The native SD21 UNet in eval mode.

unet_native_from_torchscript *Create native UNet from TorchScript*

Description

Detects architecture and loads weights from a TorchScript UNet file.

Usage

```
unet_native_from_torchscript(torchscript_path, verbose = TRUE)
```

Arguments

torchscript_path	Path to TorchScript UNet .pt file
verbose	Print loading progress

Value

A native UNet module with loaded weights

unet_safetensors *Load HF safetensors weights into the native SD/SDXL UNet*

Description

The native UNet modules mirror the diffusers `UNet2DConditionModel` state-dict keys 1:1, with the sole exception that the time- (and, for SDXL, add-) embedding MLPs are flattened from dotted to underscored names (`time_embedding.linear_1` -> `time_embedding_linear_1`). These loaders read `unet/diffusion_pytorch_model.safetensors` (single file or sharded via its `.index.json`) and copy each weight into the matching native parameter, verifying that every native parameter is filled and no key or shape is left unmatched.

Details

Reads route through the shared sharded opener, so an oversize (>2 GB) single-file checkpoint on stock CRAN safetensors surfaces the actionable "rebuild with smaller shards or install the fork" message rather than a raw 32-bit overflow.

`unet_sdxl_native` *Native SDXL UNet*

Description

Native R torch implementation of SDXL UNet2DConditionModel. SDXL has a different architecture from SD21: - 3 down/up blocks (not 4) - Variable transformer depth per block - Additional conditioning via add_embedding

Usage

```
unet_sdxl_native(in_channels = 4L, out_channels = 4L,
                 block_out_channels = c(320L, 640L, 1280L),
                 layers_per_block = 2L,
                 transformer_layers_per_block = c(0L, 2L, 10L),
                 cross_attention_dim = 2048L, attention_head_dim = 64L,
                 addition_embed_dim = 1280L, addition_time_embed_dim = 256L)
```

Arguments

`in_channels`	Input channels (default 4 for latent space)
`out_channels`	Output channels (default 4)
`block_out_channels`	Channel multipliers per block
`layers_per_block`	Number of ResBlocks per down/up block
`transformer_layers_per_block`	Transformer depth per block
`cross_attention_dim`	Context dimension from text encoder
`attention_head_dim`	Dimension per attention head
`addition_embed_dim`	Dimension for additional embeddings
`addition_time_embed_dim`	Dimension for time embedding projection

Value

An nn_module representing the SDXL UNet

`unet_sdxl_native_from_safetensors`
Build a native SDXL UNet from a diffusers safetensors directory

---

Description

The safetensors counterpart to `unet_sdxl_native_from_torchscript`: constructs `unet_sdxl_native` and loads its weights from `unet/diffusion_pytorch_model.safetensors`. Validated against the cached `stabilityai/stable-diffusion-xl-base-1.0` UNet (all 1680 parameters map with matching shapes). Pass constructor overrides through `...` for a variant checkpoint.

Usage

```
unet_sdxl_native_from_safetensors(path, verbose = TRUE, ...)
```

Arguments

`path`	Path to the UNet directory or its single-file checkpoint.
`verbose`	Print how many parameters were loaded.
`...`	Overrides for `unet_sdxl_native` constructor args.

Value

The native SDXL UNet in eval mode.

`unet_sdxl_native_from_torchscript`
Create native SDXL UNet from TorchScript

Description

Create native SDXL UNet from TorchScript

Usage

```
unet_sdxl_native_from_torchscript(torchscript_path, verbose = TRUE)
```

Arguments

`torchscript_path`	Path to TorchScript SDXL UNet .pt file
`verbose`	Print loading progress

Value

A native SDXL UNet module with loaded weights

unigram_tokenizer *Load a Unigram tokenizer from tokenizer.json*

Description

Load a Unigram tokenizer from tokenizer.json

Usage

```
unigram_tokenizer(tokenizer_path)
```

Arguments

tokenizer_path	Path to a HuggingFace tokenizer.json with a Unigram model, or a directory containing one.

Value

A `unigram_tokenizer` object.

upsampler_ltx23 *LTX-2.3 Spatial Latent Upsampler*

Description

Fresh R port of the LTX latent upsampler from the diffusers reference (Apache-2.0, pipelines/ltx2/latent_upsampler.py and pipeline_ltx2_latent_upsample.py), with the LTX 2.3 configuration: Conv3d ResBlock stages around a per-frame 2x pixel-shuffle spatial upsampler (no rational resampler). Operates on unnormalized latents.

vae_decoder_native *Native VAE Decoder*

Description

Native R torch implementation of the SDXL VAE decoder. Replaces TorchScript decoder for better GPU compatibility.

Usage

```
vae_decoder_native(latent_channels = 4, out_channels = 3,
                   block_channels = c(512, 512, 256, 128), norm_groups = 32)
```

Arguments

latent_channels	Number of latent channels (4 for SD/SDXL, 16 for FLUX/SD3)
out_channels	Number of output channels (default 3 for RGB)
block_channels	Decoder block channels (reversed encoder block_out_channels; default matches SD/SDXL and FLUX)
norm_groups	Group norm groups (default 32; must divide every entry of block_channels)

Value

An nn_module representing the VAE decoder

Examples

```
if (torch::torch_is_installed()) {
  # A small decoder; the SD/SDXL defaults are far too large to build
  # inside an example.
  decoder <- vae_decoder_native(latent_channels = 4,
                                block_channels = 32,
                                norm_groups = 32)
  latents <- torch::torch_randn(c(1, 4, 8, 8))
  image <- torch::with_no_grad(decoder(latents))
  image$shape
}

# Real weights come from a downloaded checkpoint.
## Not run:
decoder <- vae_decoder_native()
load_decoder_weights(decoder, "path/to/decoder.pt")

## End(Not run)
```

---

vae_decoder_native_from_safetensors

Build a native VAE decoder from a diffusers safetensors directory

---

Description

The safetensors counterpart to the TorchScript decoder path: constructs vae_decoder_native and loads the decoder half of a diffusers AutoencoderKL checkpoint (no TorchScript, so it works on Blackwell). latent_channels defaults to 4 (SD/SDXL); pass 16 for the FLUX/SD3 VAE. The SD/SDXL and FLUX VAEs share the decoder shape and differ only in that channel count.

Usage

```
vae_decoder_native_from_safetensors(path, latent_channels = 4L, verbose = TRUE,
                                    ...)
```

Arguments

path	Path to the VAE directory (containing diffusion_pytorch_model.safetensors) or the file itself.
latent_channels	Latent channel count (4 for SD/SDXL, 16 for FLUX).
verbose	Print how many parameters were loaded.
...	Overrides for vae_decoder_native constructor args.

Value

The native VAE decoder in eval mode.

---

vae_flux2 *FLUX.2 Latent Layout and VAE Helpers*

Description

Fresh R port of the FLUX.2 latent packing chain from the diffusers reference (Apache-2.0, src/diffusers/pipelines/flux2/pipeline_flux2_klein.py). The 32-channel VAE latent is patchified 2x2 into 128 channels, normalized with the VAE's BatchNorm running statistics (there is no scalar scaling/shift factor in FLUX.2), and flattened to channels-last tokens for the transformer.

---

vae_ltx23 *LTX-2.3 Causal Video VAE*

Description

Fresh R port of the LTX-2 video autoencoder from the diffusers reference (Apache-2.0, autoencoder_kl_ltx2.py), with LTX 2.3 defaults: encoder blocks (256, 512, 1024, 1024), a 4-up-block decoder with mixed (spatiotemporal, spatiotemporal, temporal, spatial) upsampling, no upsample residuals, and zeros spatial padding throughout. The encoder is causal; the decoder is not.

---

vae_ltx23_modules *LTX-2.3 Video VAE Building Blocks*

Description

Fresh R port of the LTX-2 causal video autoencoder blocks from the diffusers reference (Apache-2.0, src/diffusers/models/autoencoders/autoencoder_kl_ltx2.py). Training and unused inference branches (noise injection, timestep conditioning, plain-conv downsampling) are intentionally not ported; the 2.3 checkpoints carry no such weights.

vocab_size *Get vocabulary size*

Description

Get vocabulary size

Usage

```
vocab_size(tokenizer)
```

Arguments

tokenizer	A bpe_tokenizer object.

Value

Integer vocabulary size.

vocoder_ltx23 *LTX-2.3 Vocoder with Bandwidth Extension*

Description

Fresh R port of the LTX-2 BigVGAN-style vocoder from the diffusers reference (Apache-2.0, pipelines/ltx2/vocoder.py). The 2.3 vocoder runs a 16 kHz stage (hidden 1536, snakebeta activations with anti-aliased up/downsampling), re-analyzes its output into a causal log-mel spectrogram, and feeds a bandwidth-extension vocoder whose residual is added to a Hann-resampled skip path for 48 kHz output. The Kaiser sinc / Hann filters and STFT bases are checkpoint buffers. Runs in float32 (small model; snakebeta is precision-sensitive).

vram *VRAM Detection and Management Utilities*

Description

Device detection, VRAM reporting, and module offloading helpers shared by the image and video pipelines.

vram_report *Report VRAM Usage*

Description

Prints current VRAM usage from nvidia-smi.

Usage

```
vram_report(label = "")
```

Arguments

label	Character. Label for the report.

Value

Invisibly returns a list with used and free VRAM in GB.

Examples

```
if (torch::torch_is_installed()) {
  vram_report("After model load")
}
```

write_wav *Write a 16-bit PCM WAV file*

Description

Minimal RIFF writer in base R.

Usage

```
write_wav(audio, path, sample_rate = 48000L)
```

Arguments

audio	Numeric matrix [channels, samples] in [-1, 1].
path	Output path.
sample_rate	Integer.

Value

Invisibly, the path.

zimage_block *Z-Image transformer block*

Description

Sandwich-norm residual block shared by the noise refiner, the context refiner and the main trunk. With `modulation = TRUE` the block carries an adaLN linear producing (scale_msa, gate_msa, scale_mlp, gate_mlp); the context refiner uses `modulation = FALSE` and has no adaLN weights at all.

Usage

```
zimage_block(dim, n_heads, norm_eps = 1e-05, modulation = TRUE)
```

Arguments

`dim`	Integer. Model width.
`n_heads`	Integer. Attention heads; head dim is dim / n_heads.
`norm_eps`	Numeric. RMSNorm epsilon. Default 1e-5.
`modulation`	Logical. Whether the block is timestep-modulated.

Value

Module whose forward(x, freqs, adaln_input, chunk_size) returns the residual block output, a tensor of the same shape as `x`. `adaln_input` is used only when the block was built with `modulation = TRUE`.

zimage_cap_pos_ids *Build Z-Image caption position ids*

Description

Caption tokens ramp 1..cap_padded_len on the first axis (axes 2 and 3 zero). The reference builds the grid over the already-padded length, so pad tokens continue the ramp rather than sitting at the origin (the (0,0,0) pad ids it also emits are truncated away in _prepare_sequence and never reach RoPE).

Usage

```
zimage_cap_pos_ids(cap_padded_len, device = "cpu")
```

Arguments

`cap_padded_len`	Integer. Caption length after padding to a multiple of 32.
`device`	Device for the resulting tensor.

Value

Float tensor of shape [cap_padded_len, 3].

---

zimage_feed_forward *Z-Image feed-forward (SwiGLU with separate gate weights)*

Description

w2(silu(w1(x)) * w3(x)) with all three linears bias-free. The hidden width is int(dim / 3 * 8).

Usage

```
zimage_feed_forward(dim, hidden_dim)
```

Arguments

dim	Integer. Model width.
hidden_dim	Integer. Hidden width.

Value

Module whose forward(x) returns `w2(silu(w1(x)) * w3(x))`, a tensor of the same shape as `x`.

---

zimage_final_layer *Z-Image final layer*

Description

Parameterless LayerNorm scaled by 1 + adaLN(c) (scale only, no shift), then the token-to-patch projection.

Usage

```
zimage_final_layer(hidden_size, out_channels)
```

Arguments

hidden_size	Integer. Model width.
out_channels	Integer. Patch output dim (patch^2 * f_patch * latent channels).

Value

Module whose forward(x, c) returns the token-to-patch projection [B, S, out_channels], ready for unpatchifying into a latent.

zimage_img_pos_ids *Build Z-Image latent image position ids*

Description

Image tokens use axis 1 for the frame index offset past the caption (start0 = cap_padded_len + 1), axis 2 for the token row and axis 3 for the token column. Trailing pad tokens (token count not a multiple of 32) sit at (0, 0, 0). Reference: patchify_and_embed / _pad_with_ids.

Usage

```
zimage_img_pos_ids(h_tokens, w_tokens, start0, f_tokens = 1L, device = "cpu")
```

Arguments

h_tokens	Integer. Token grid height (latent height / patch).
w_tokens	Integer. Token grid width (latent width / patch).
start0	Integer. First-axis start, cap_padded_len + 1.
f_tokens	Integer. Token grid frames; 1 for txt2img.
device	Device for the resulting tensor.

Value

Float tensor of shape [padded token count, 3].

zimage_is_quant_key *Test whether a Z-Image key is in the quantization cast set*

Description

Test whether a Z-Image key is in the quantization cast set

Usage

```
zimage_is_quant_key(key)
```

Arguments

key	Character vector of parameter names (diffusers-style).

Value

Logical vector.

`zimage_load_pipeline` *Load the Z-Image-Turbo pipeline*

Description

Loads the quantized transformer artifact plus the 16-channel VAE decoder, Qwen3-4B text encoder, and tokenizer from the HuggingFace cache populated by `download_zimage_turbo`. With fp8 precision the ~6.3 GB transformer rides to the GPU per phase.

Usage

```
zimage_load_pipeline(model_dir = NULL, device = "cuda",
                     precision = c("auto", "fp8", "nf4", "bf16"),
                     text_device = NULL, attn_chunk = NULL,
                     phase_offload = TRUE, pin = NULL, verbose = TRUE)
```

Arguments

`model_dir`	Quantized artifact directory (default: the `download_zimage_turbo` location for `precision`), or a raw diffusers transformer directory.
`device`	Character. Compute device.
`precision`	"auto" (default: reuse an existing artifact, else fp8 when safetensors supports float8, else nf4), "fp8", or "nf4".
`text_device`	Device for the Qwen3 encoder (default: `device`; it encodes in its own phase and offloads).
`attn_chunk`	Integer or NULL. Attention query-chunk override.
`phase_offload`	Logical. One GPU tenant per phase.
`pin`	Logical or NULL. Page-lock the phase-swapped weights for DMA-rate transfer (see `staging`). NULL (default) resolves via `options(diffuseR.pin_staging)` then the host-RAM-aware `recommend` decision.
`verbose`	Logical.

Value

A `zimage_pipeline` list.

zimage_patchify *Patchify a latent image to Z-Image tokens*

Description

(C, F, H, W) -> [F/pF * H/p * W/p, pF * p * p * C], matching _patchify_image. No padding is applied here.

Usage

```
zimage_patchify(image, patch_size = 2L, f_patch_size = 1L)
```

Arguments

image	Tensor of shape [C, F, H, W].
patch_size	Integer spatial patch size. Default 2.
f_patch_size	Integer temporal patch size. Default 1.

Value

Tensor of shape [num_tokens, patch_dim].

zimage_pos_embed *Compute Z-Image rotary frequencies from position ids*

Description

Per-axis 1D rotary frequencies in the interleaved-real convention. Frequencies and angles are built in float64, then the angles are cast to float32 before cos/sin — matching the reference torch.polar call on a .float() tensor. Output format matches `flux_pos_embed` so `flux_apply_rotary_emb` applies unchanged.

Usage

```
zimage_pos_embed(ids, axes_dim = c(32L, 48L, 48L), theta = 256)
```

Arguments

ids	Tensor of shape [S, 3] from `zimage_cap_pos_ids` / `zimage_img_pos_ids`.
axes_dim	Integer vector of per-axis rotary dims; must sum to the attention head dim. Z-Image uses c(32, 48, 48).
theta	Numeric. RoPE base frequency. Z-Image uses 256.

Value

List of two tensors (cos, sin), each [S, sum(axes_dim)], float32, on the device of `ids`.

`zimage_transformer` *Z-Image Transformer*

Description

Fresh R port of ZImageTransformer2DModel from the diffusers reference (Apache-2.0, src/diffusers/models/transformers/tra
Single-stream DiT: image tokens pass through a modulated noise refiner, caption tokens through an unmodulated context refiner, then both are concatenated (image first) and run through the main trunk. The module tree mirrors the reference state-dict keys 1:1 (all_x_embedder.2-1, noise_refiner.N, context_refiner.N, layers.N, all_final_layer.2-1, t_embedder, cap_embedder, x_pad_token, cap_pad_token).

Usage

```
zimage_transformer(in_channels = 16L, dim = 3840L, n_layers = 30L,
                   n_refiner_layers = 2L, n_heads = 30L, norm_eps = 1e-05,
                   cap_feat_dim = 2560L, rope_theta = 256, t_scale = 1000,
                   axes_dims = c(32L, 48L, 48L), patch_size = 2L,
                   f_patch_size = 1L)
```

Arguments

`in_channels`	Integer. Latent channels. Default 16.
`dim`	Integer. Model width. Default 3840.
`n_layers`	Integer. Main trunk depth. Default 30.
`n_refiner_layers`	Integer. Refiner depth. Default 2.
`n_heads`	Integer. Attention heads. Default 30.
`norm_eps`	Numeric. RMSNorm epsilon. Default 1e-5.
`cap_feat_dim`	Integer. Caption embedding width. Default 2560.
`rope_theta`	Numeric. RoPE base frequency. Default 256.
`t_scale`	Numeric. Timestep scale. Default 1000.
`axes_dims`	Integer vector. Per-axis rotary dims. Default c(32, 48, 48).
`patch_size`	Integer. Spatial patch size. Default 2.
`f_patch_size`	Integer. Temporal patch size. Default 1.

Details

This port is batch-of-1: `x` is a single latent [C, F, H, W] and `cap_feats` a single caption [L, cap_feat_dim], so sub-sequences are uniform and no attention mask is needed. Padding to a multiple of 32 tokens uses the learned pad parameters, appended after embedding (the reference pads raw features with repeats, embeds pointwise, then overwrites the pad rows with the same learned tokens).

Value

Module whose forward(x, t, cap_feats, chunk_size) returns the predicted velocity for the single latent, a tensor [C, F, H, W] matching the shape of `x`. Note that the checkpoint negates this output and consumes a reversed timestep; see `txt2img_zimage`.

`zimage_t_embedder` *Z-Image timestep embedder*

Description

256-dim cos-first sinusoid (computed in float32) through a Linear-SiLU-Linear MLP. The model feeds t * t_scale with the pipeline's t already in [0, 1].

Usage

```
zimage_t_embedder(out_size, mid_size = 1024L, freq_size = 256L)
```

Arguments

`out_size`	Integer. Output width, min(dim, 256).
`mid_size`	Integer. Hidden width. The full model uses 1024.
`freq_size`	Integer. Sinusoid width. Default 256.

Value

Module whose forward(t) returns the timestep embedding [B, out_size].

`zimage_unpatchify` *Unpatchify Z-Image tokens back to a latent image*

Description

Takes the first F/pF * H/p * W/p tokens (the image span of the unified sequence) and reassembles [C, F, H, W], matching unpatchify.

Usage

```
zimage_unpatchify(tokens, size, patch_size = 2L, f_patch_size = 1L,
                  out_channels = 16L)
```

Arguments

tokens	Tensor of shape [S, pF * p * p * C] with the image tokens first.
size	Integer vector c(F, H, W) of the target latent size.
patch_size	Integer spatial patch size. Default 2.
f_patch_size	Integer temporal patch size. Default 1.
out_channels	Integer number of latent channels. Default 16.

Value

Tensor of shape [C, F, H, W].

Index