hrm_utils — Python API Reference

Minimal loader + model implementation for HRM (Hierarchical Reasoning Model). Supports single or sharded checkpoints, attention-key adaptation to PyTorch MHA, config inference, vocab-size safety, optional tokenizer loading/resizing, and reliable embedding tying.

Quickstart

```
from hrm_utils import load_hrm

model, tokenizer = load_hrm(
    "path/to/checkpoint_or_dir",
    device="cuda",            # "cpu" | "mps" | "cuda" | "auto"
    dtype="auto",             # "fp32" | "bf16" | "fp16" | "auto"
    strict=False,             # allow benign key diffs
    with_tokenizer=True,      # try to load local tokenizer from same dir
)
model.eval()

# simple greedy loop (example)
import torch
ids = tokenizer.encode("Hello", return_tensors="pt").to(model.lm_head.weight.device)
for _ in range(30):
    out = model(ids)
    next_id = out["logits"][:, -1, :].argmax(-1, keepdim=True)
    ids = torch.cat([ids, next_id], dim=1)
print(tokenizer.decode(ids[0].tolist()))
```

Top-Level API

load_hrm

```
load_hrm(
    checkpoint_or_dir: str,
    device: Optional[str] = "auto",
    dtype: str = "auto",
    strict: bool = True,
    override_config: Optional[Dict[str, Any]] = None,
    ModelCls=None,
    with_tokenizer: bool = False,
    tokenizer_path: Optional[str] = None,
) -> HRMForCausalLM | Tuple[HRMForCausalLM, "PreTrainedTokenizerFast"]
```

Description Load an HRM model from a file or directory, adapting state-dict keys as needed. Optionally load a local tokenizer and automatically resize/tie embeddings.

checkpoint_or_dir (str): Path to a checkpoint file or a directory containing a checkpoint. Supports single files (*.safetensors, pytorch_model.bin, model.bin, model.pt) and sharded indices (model.safetensors.index.json, pytorch_model.bin.index.json).

device (str|None): "auto" (default; prefers CUDA → MPS → CPU), or explicit "cuda"|"mps"|"cpu".

dtype (str): "auto" (bf16 on supported CUDA, else fp32), or "fp32" | "bf16" | "fp16".

strict (bool): If True, raise when keys are missing/unexpected after adaptation; if False, print a summary and continue. Default True.

override_config (dict|None): Extra/override config fields. Recognized: vocab_size, d_model, n_heads, d_ff, dropout, k_l_steps, max_cycles, ponder_loss_weight.

ModelCls (type|None): Alternate model class compatible with HRMForCausalLM constructor/signature.

with_tokenizer (bool): If True, try loading a local tokenizer from tokenizer_path (or the checkpoint dir).

tokenizer_path (str|None): Directory containing tokenizer files (defaults to the checkpoint directory).

Returns • HRMForCausalLM (if with_tokenizer=False) • (HRMForCausalLM, tokenizer) if with_tokenizer=True (tokenizer may be None).

Behavior & Notes

- Config inference: If config.json is absent, config is inferred from weights (embedding and MLP dims).
- Vocab size safety: Prefers vocab_size from the checkpoint tensors to avoid size mismatches.
- Attention key adaptation: Converts qkv.* or split q/k/v projections into PyTorch MHA in_proj_* format.
- Tied-weights checkpoints: If lm_head.weight is missing but tok_emb.weight exists, it is mirrored to avoid a warning.
- Dtype move: Parameters and buffers are moved to target device and dtype.
- Tokenizer: Local-only load; AutoTokenizer → tokenizer.json → vocab.json+merges.txt. If loaded, embeddings are resized and re-tied.

HRMForCausalLM

```
class HRMForCausalLM(nn.Module):
    def __init__(
        self,
        vocab_size: int,
        d_model: int = 512,
        n_heads: int = 8,
        d_ff: int = 2048,
        dropout: float = 0.1,
        k_l_steps: int = 4,
        max_cycles: int = 8,
        ponder_loss_weight: float = 1e-2,
    )
```

Hierarchical Reasoning Model with alternating L/H attention blocks and a learned halting head (ponder cost). Input and output embeddings are tied.

vocab_size (int): Vocabulary size.

d_model (int): Hidden size / MHA model dimension. Must be divisible by n_heads.

n_heads (int): Number of attention heads.

d_ff (int): Feed-forward (SwiGLu) hidden size.

dropout (float): Dropout prob used in attention and MLP.

k_l_steps (int): Count of L-module refinement steps per cycle.

max_cycles (int): Max L/H cycles before forced halt.

ponder_loss_weight (float): Weight of the halting/ponder cost added to LM loss.

Attributes

- tok_emb (nn.Embedding): Token embedding (tied to lm_head.weight).
- pos_emb (SinusoidalPositionalEmbedding): Fixed sinusoidal positions.
- L_mod, H_mod (AttnBlock): Alternating attention blocks.
- halt_head (nn.Linear): Scalar halt probability per token step.
- out_norm (RMSNorm): Output norm before lm_head.
- lm_head (nn.Linear): Language modeling head with tied weights.

.forward

```
forward(
    input_ids: torch.LongTensor,          # shape: [B, L]
    attention_mask: Optional[torch.BoolTensor] = None,  # shape: [B, L]
    labels: Optional[torch.LongTensor] = None,          # shape: [B, L]
) -> Dict[str, torch.Tensor]
```

Inputs input_ids [B,L]; attention_mask [B,L] (1/True = real tokens, 0/False = pad); labels [B,L] for teacher forcing.

Returns dict with keys: 'logits' [B,L,V], 'loss' (optional), 'lm_loss' (optional), 'ponder_loss' (optional).

Details Causal mask is upper-triangular bool (True = future masked). Loop breaks when all tokens' remaining mass < 1e-4. Positional embeddings share dtype with token embeddings.

Interop helpers

```
get_input_embeddings() -> nn.Embedding
set_input_embeddings(new_emb: nn.Embedding) -> None  # re-ties lm_head
tie_weights() -> None
```

Building Blocks (advanced)

AttnBlock

```
AttnBlock(d_model: int, n_heads: int, d_ff: int, pdrop: float = 0.1)
forward(
  x: torch.Tensor,                      # [B, L, d_model]
  attn_mask: Optional[torch.BoolTensor] = None,         # [L, L] causal mask (True = masked)
  key_padding_mask: Optional[torch.BoolTensor] = None,  # [B, L] (True = pad/ignore)
) -> torch.Tensor
```

Residual MHA + RMSNorm + SwiGLU block (batch_first).

RMSNorm

```
RMSNorm(d: int, eps: float = 1e-6)
forward(x: torch.Tensor) -> torch.Tensor
```

Root-mean-square normalization with learned scale.

SwiGLU

```
SwiGLU(d_model: int, d_ff: int, pdrop: float = 0.1)
forward(x: torch.Tensor) -> torch.Tensor
```

SwiGLU MLP with dropout.

SinusoidalPositionalEmbedding

```
SinusoidalPositionalEmbedding(d_model: int, max_len: int = 8192)
forward(L: int) -> torch.FloatTensor  # [1, L, d_model]
```

Fixed sinusoidal positions. Buffer is registered with persistent=False to avoid checkpoint bloat.

Checkpoint Discovery & Key Adaptation

Discovery priority (when a directory is passed):

- One *.safetensors file → use it.
- Multiple *.safetensors → prefer model.safetensors / pytorch_model.safetensors, else first sorted.
- Sharded index: model.safetensors.index.json or pytorch_model.bin.index.json (preferred over plain .bin).
- Plain files: pytorch_model.bin, model.bin, model.pt.
- Any other *.pt / *.bin.

Attention keys mapped to PyTorch MHA:

- Fused qkv: {prefix}.qkv.(weight|bias) → {prefix}.in_proj_(weight|bias)
- Split projections: {prefix}.{q,k,v}_proj.(weight|bias) → concat along dim 0 → in_proj_(weight|bias)
- Output: {prefix}.o.(weight|bias) → {prefix}.out_proj.(weight|bias)
- Missing biases created as zeros when the receiving module expects them.

Tokenizer Loading

- AutoTokenizer.from_pretrained(local, use_fast=True, local_files_only=True, trust_remote_code=True)
- Fallback: tokenizer.json → PreTrainedTokenizerFast
- Fallback: vocab.json + merges.txt → GPT2TokenizerFast (+ special_tokens_map.json if present)
- If loaded and size differs from model, embeddings are resized and re-tied.

Recipes

```
@torch.no_grad()
def greedy_generate(model, tokenizer, prompt: str, max_new_tokens=64, temperature=1.0):
    dev = next(model.parameters()).device
    ids = tokenizer(prompt, return_tensors="pt").input_ids.to(dev)
    for _ in range(max_new_tokens):
        logits = model(ids)["logits"][:, -1, :] / max(temperature, 1e-6)
        next_id = torch.argmax(logits, dim=-1, keepdim=True)
        ids = torch.cat([ids, next_id], dim=1)
    return tokenizer.decode(ids[0].tolist())
```

Mixed precision inference (bf16 on Ampere+):

```
model, tok = load_hrm("ckpt_dir", device="cuda", dtype="bf16", with_tokenizer=True, strict=False)
```

Tied-weights checkpoints: when only tok_emb.weight is saved, the loader mirrors it as lm_head.weight automatically.