Reading the Architecture off the Weights: Manifest-Only Recovery of Neural Network Anatomy

Sattaya Singkul
Researcher
joeysattaya@gmail.com

Abstract

Engineers who deploy a model need to know what it is made of: which blocks repeat, what each block computes, how the parts connect and where the parameters sit. This information is usually spread over model cards, family-specific configuration files and modeling code. We show that most of it can be read from the tensor manifest alone, the list of tensor names, shapes and storage types that a safetensors file keeps in a header of a few tens of kilobytes. Our method normalizes quantized storage, finds repeated blocks in the module tree, assigns roles to the parts of each block from the algebra of their shapes, and links components by matching their widths. Fields that shapes cannot determine, such as the number of attention heads, are read from `config.json` when it exists, and the configuration is checked against the weights. On 510 architectures from the `transformers` library, with ground truth taken from module classes that the method never sees, per-tensor role accuracy is 96.8%, against 82.3% for the strongest name-rule baseline, and 95.0% when every name is replaced by a random token. Parameter counts are exact for every model, including under NVFP4, FP8 and GPTQ storage. On 191 popular Hugging Face checkpoints read over HTTP range requests, totals agree with the Hub's own count wherever the Hub counts logical parameters, and configuration fusion recovers head geometry for 96 to 98% of models. The method runs in about 10 ms for a typical model, and we release it as an open explorer that reads any Hugging Face repository without downloading weights.

1 Introduction

Anyone handed a new checkpoint asks the same questions. How many layers does it have, and are they all alike? Is it a mixture of experts, and how much of its size is experts? Which part runs once per input and which part runs once per generated token? For a well-known family, `config.json` answers these questions. For anything new it often does not. Keys differ between families (`hidden_size`, `d_model`, `n_embd`), composite models nest several schemas, and fields such as `layers_block_type` only make sense after reading the modeling code. Reading that code is slow, and running it through `trust_remote_code` is a known risk on public hubs [32]. Graph viewers such as Netron [23] or Model Explorer [8] need an exported graph and, in practice, the whole file.

A much smaller description is almost always available. A safetensors file [14] starts with an 8-byte length followed by a JSON header that lists the name, dtype, shape and byte range of every tensor. Two HTTP range requests are enough to fetch it. Across the 510 architectures studied here its median size is 46 KB, about 6×10^{-5} of the weights in BF16. PyTorch zip checkpoints keep the same information in their `data.pkl` record, which can be read without touching tensor data.

This paper asks how much of a model's architecture can be recovered from the manifest alone, and how reliably. The question is practical: a tool that answers it can explain a hub model in a browser tab before anyone downloads it. It also has an analytical side, because some properties leave no trace in tensor shapes.

An honest tool should say which ones, measure how often a default guess for them is right, and take them from a better source when one exists.

Our starting point is that a manifest carries two kinds of evidence. The name tree shows structure: integer-indexed lists are stacks of blocks, identical subtrees are identical block types, and the module that owns a stack marks a component. The shapes show function. All blocks in a stack share a residual width d. An attention region is a set of projections that leave d and one that returns to it. A mixture of experts is a set of tensors whose leading dimension equals the output size of a router. A selective state-space mixer is a depthwise convolution next to a few per-head vectors. Two components are connected where the output width of one equals the input width of the other. Names help to break ties, but, as Section 6 shows, most of the work survives when every name is replaced by a random token.

The paper makes three contributions. First, we define manifest-only architecture recovery, give a representation of a model's anatomy that is useful to engineers, and analyse which of its properties shapes cannot identify (Section 3.1). Second, we describe a recovery algorithm that combines module-tree structure with shape algebra and width matching, handles six quantized storage layouts, and fuses non-identifiable fields from the configuration while checking it against the weights (Section 3). Third, we evaluate it on 510 architectures with independent ground truth and on 191 popular checkpoints read directly from the Hugging Face Hub and release it as a bilingual web tool (Sections 4–7). The appendix lists the result for every model.

2 Related work

A checkpoint usually comes with three descriptions of the same model: the weights, a configuration file and code. Model cards [17] add a human summary whose architectural fields are free text. The configuration is authoritative for the library that wrote it, but its schema depends on the family and, for new families, has to be interpreted through code. Pickle-based weight formats can execute code when loaded [32]. Safetensors avoids this, and its header is the most uniform self-description that exists across families. Our work treats that header as the primary source.

The closest existing practice is the name tables that conversion tools maintain, such as llama.cpp's mapping from Hugging Face module paths to its own tensor types [7]; they are precise for the architectures they list and silent elsewhere, and we use one as a baseline. Netron [23], TensorBoard and Model Explorer [8] draw operator graphs from exported formats. They show a graph faithfully but do not group operators into semantic parts, and they need the full file. Transformer Explainer [2] explains one fixed architecture interactively. Our tool borrows its step-by-step style but derives the content from the manifest, so it applies to any architecture.

Architecture recovery has been studied mostly as an attack. Side-channel methods reconstruct a network from cache timing or memory traces of a running model [13, 29]; in that setting the observer has no access to the file. Here the header is public, and the difficulty lies in interpreting it. Weight fingerprinting such as HuRef [31] identifies a model's lineage from weight values, which we never read.

The shape rules encode the signatures of attention and its variants, including grouped-query [1] and latent attention [4], sparse mixtures of experts [5, 25], selective state-space and linear-attention mixers [3, 10, 30] and their hybrids [18], conformers [11, 22], transducers [9, 28], diffusion transformers [20] and tabular in-context learners [12, 21]. The storage rules cover GPTQ [6], AWQ [15], FP8 [16], MXFP4 [24], NVFP4 [19] and the MLX and bitsandbytes layouts.

3 Method

This section defines the problem, gives an overview of the system, and then describes the recovery algorithm and the configuration fusion step in turn.

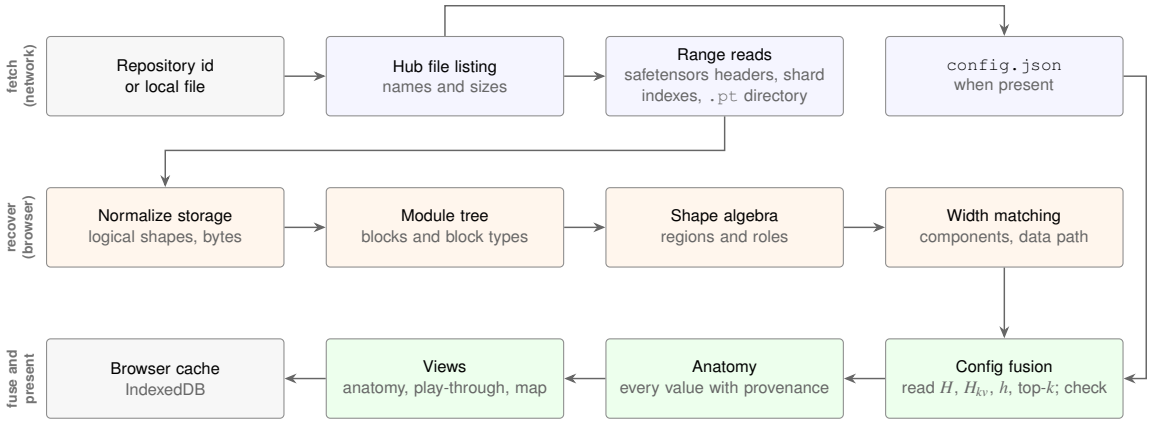


Figure 1: Workflow. Only descriptive bytes are fetched: the file listing, safetensors headers, shard indexes, the zip directory and pickle record of PyTorch checkpoints, and the configuration. Recovery and fusion run in the browser, and results are cached locally.

3.1 Problem formulation

A manifest is a set $\mathscr{M} = \{(n_i, s_i, \tau_i)\}$ of tensor names (dot-separated paths), shapes and storage dtypes. A configuration $\mathscr{C}$, a nested key–value tree, may also be available. The goal is an anatomy $\mathscr{A}$ with four levels. At the top are components, such as a vision tower, a projector, a language backbone or a prediction network, each with a residual width d, a modality and a role, and connected by directed edges that form the data path. Each component contains stacks of repeated blocks, and each block is assigned a block type. Each block type is divided into regions, labelled with one of nine roles: attention, cross-attention, feed-forward, mixture of experts, SSM or linear attention, convolution module, adaLN modulation, recurrent cell and norm. Tensors outside blocks are labelled input stem, output head or adapter. Finally, the anatomy carries logical shapes under quantized storage and derived quantities: parameters, stored bytes, KV-cache bytes per token and active parameters per token. Every architectural field records where it came from: measured from the manifest, read from $\mathscr{C}$ and checked against the manifest, or estimated.

3.2 What a manifest cannot identify

Some properties leave no trace in shapes. The most important is the split of attention width into heads. A projection $W_q \in \mathbb{R}^{Hh \times d}$ with H heads of size h has the same shape as one with $H/2$ heads of size $2h$, so only the product Hh and the ratio of query to key–value width are visible. The pair becomes identifiable only through per-head parameters, such as query and key norms of width h or relative-position biases of shape $H \times h$. Masking is also invisible: a decoder-only language model with tied embeddings stores exactly the tensors of a bidirectional encoder of the same size. The number of experts is visible but the number used per token is not. Gated and ungated feed-forwards differ in the number of matrices, but SiLU and GELU do not differ at all, and pre-norm and post-norm blocks store identical tensors.

These limits shape the design. The algorithm reports such fields only as estimates, the interface labels them as estimates, and when a configuration exists the fields are read from it and checked against the shapes (Section 3.5). The evaluation measures both how often the estimate is right and how much the configuration adds.

Algorithm 1 Manifest-only anatomy recovery with configuration fusion

Require: manifest $\mathscr{M}$, optional configuration $\mathscr{C}$
$\mathscr{T} \leftarrow$ NORMALIZE($\mathscr{M}$) ▷ logical shapes, stored bytes
build the module tree; find lists, expert lists, sub-layer lists, block lists and their owners
for all block list L **do**
    group items by structural signature into block types
    for all block type β **do**
        $d \leftarrow$ width of the leaf norms of a representative block
        classify its children by the rules of Table 1; split nested blocks; pair halves
    end for
end for
label stems and heads; find adapters, transducer joints, timestep embedders, message passing
choose the backbone and the family; derive d, L, f, E, V and KV bytes per token
if $\mathscr{C}$ is given **then**
    match sub-configurations to components by width and depth; read H, H_{kv}, h, top-k
    check $Hh = x$ and $H_{kv}h = x_{kv}$; mark each field as read from $\mathscr{C}$ or report the mismatch
end if
return the anatomy, with provenance and evidence for every decision

3.3 System overview

Figure 1 shows the workflow. Given a repository id, the explorer lists the repository and decides which files describe the weights. Safetensors files are preferred. Their shard indexes are read per folder, so a diffusers repository with `transformer/`, `text_encoder/` and `vae/` folders yields one name prefix per component. For each safetensors file, two range requests fetch the length and the header; up to eight files are read in parallel. If a repository only has PyTorch zip checkpoints, the reader fetches the last 64 KB of the file to find the zip central directory, then fetches only the `data.pkl` record and interprets it with a restricted unpickler that rebuilds names, shapes and dtypes without resolving any Python callable. The configuration is fetched alongside. The manifest then passes through the recovery algorithm of Section 3.4 and the result through configuration fusion (Section 3.5). Each step is shown to the user as it happens ("reading header 3 of 12, 41 KB so far"), and the final anatomy is cached in the browser.

3.4 Recovery algorithm

The rules in this subsection read only shapes, dtypes and the structure of the name tree. A small lexicon of name fragments (`attn`, `mlp`, `experts` and similar) breaks ties that shapes leave open. It can be switched off, and Section 6 reports results with and without it. Algorithm 1 summarizes the procedure.

3.4.1 Normalizing storage

Quantized checkpoints store one logical weight as several tensors. We group tensors by module and recognize the layout from dtypes and shape ratios before looking at names. NVFP4 stores a `U8` tensor of shape $[o, i/2]$, an FP8 scale tensor $[o, i/16]$ and a scalar second-level scale, so the logical shape is $[o, i]$ and the group size follows from the ratio. FP8 stores an FP8 weight with a scalar, per-channel $[o, 1]$ or block scale. GPTQ stores an `I32` tensor $[i/8, o]$ with scales $[i/g, o]$; AWQ packs the other axis, and the two are told apart by the axis the scales share. MXFP4, MLX and bitsandbytes layouts follow the same pattern. Auxiliary tensors count toward stored bytes but not toward parameters.

Role	Shape signature inside a block of residual width d
Mixture of experts	stacked experts $[E,\cdot,\cdot]$ or an expert list, with a router $[E',d]$, $E \le E' \le E+8$
SSM / linear attn.	depthwise conv $[C,1,k]$, $k \le 8$, with recurrence parameters on the mixer: two or more vectors of one length $n_h \ne d$, or a state tensor with a small last dimension; without a conv, two or more per-head gates $[\le 64,d]$ or decay vectors
Feed-forward	one output $[d,f]$ and one or two inputs $[f,d]$, or a fused input $[2f,d]$ with $f > 1.3d$
Attention	one output $[d,x]$ and inputs from d, or a fused input $[x+2x_{kv},d]$; low-rank chains that stay inside the region count as internal
Cross-attention	as attention, with at least two projections from another component's width $w \ge 128$
Recurrent cell	two or more matrices with $3h$ or $4h$ rows (GRU, LSTM), none returning to d
Convolution module	depthwise conv with pointwise $[C,C',1]$ convs or batch-norm statistics
adaLN modulation	one matrix $[kd,t]$ from a conditioning width $32 \le t < d$, or $k \in \{6,9,12,18\}$ when $t = d$, or a $[1,k,d]$ table

Table 1: Region signatures. A matrix $[o,i]$ is an input to the region if $i = d$ and an output if $o = d$.

3.4.2 Blocks from the module tree

Names form a tree. A node whose children are all integers, or keys such as `expert_0`, `expert_1`, is a list. An item is composite if its subtree holds at least two modules with parameters; this separates blocks from `nn.Sequential` wrappers around single layers. Expert lists are lists inside an item of another list whose items are identical two- or three-matrix MLPs without norms, usually beside a router whose first dimension equals the number of items. Stacked experts appear instead as 3-D tensors $[E,\cdot,\cdot]$ with one trailing dimension equal to the block width. Block lists are the innermost composite lists that are not expert lists, with one correction for sub-layers: when every item of an outer list holds short inner lists of equal length, as in T5's `block.`*i*`.layer.`*j* or LongCat's two attention and two MLP sub-layers per layer, the inner items are sub-layers and the outer items are the blocks.

Each block list belongs to the module that owns it, raised through its ancestors while no other stack lives there and, in multi-folder repositories, never above the folder. These owners become components, so that embeddings and heads next to an `encoder` wrapper join the right component. Within a list, each block gets a signature, the sorted multiset of (relative path with indices masked, tensor rank) pairs, and identical signatures form a block type. In a hybrid stack such as Nemotron-H [18] this gives one type each for the Mamba-2, MoE and attention layers, and the sequence of types is the layer pattern.

3.4.3 Regions by shape algebra

For each block type we analyse one representative block. Its residual width d is the length of the vectors held by its leaf norm modules. This is more robust than counting matrix dimensions, which the inner width of an SSM can dominate. The children of the block are then classified with the signatures in Table 1. For attention, vectors in norm-only submodules whose length divides x give the head size, and parameters of shape $[H,h]$ with $Hh = x$ give both factors.

Four refinements matter in practice. A child that holds two sub-modules of different confident roles, such as an attention and an MLP inside one "inducing block", is split. Two half feed-forwards in sibling modules ($[f,d]$ in `intermediate` and $[d,f]$ in `output`) are paired. If a region is weak in the usual layout but becomes attention when every matrix is transposed, the weights are stored as $[\text{in},\text{out}]$, as GPT-2's `Conv1D` layers are. Finally, a list whose items map d to another width and never return to d is not a residual stack: its items are read as per-layer prediction heads, such as the box MLPs of DETR-style detectors, or as projectors when the output width belongs to another component, as with the DeepStack mergers of Qwen3-VL.

3.4.4 Stems, heads and the data path

Outside blocks, a matrix $[V, d]$ with $V > d$ is a token table and its twin an untied output head. A vector of length V is the head's bias, and a small table beside a token table or a convolutional stem is a position or type embedding. A chain of matrices whose free input width equals the width d_X of one component, or $k^2 d_X$ for $k \in \{2, 3, 4\}$ after pixel shuffling, and whose free output width equals the width d_Y of another, becomes an adapter with edges $X \to$ adapter $\to Y$.

Three further patterns become components of their own. A transducer has a label embedding $[V, h_p]$ next to a recurrent cell, an encoder projection $[j, d_{\text{enc}}]$ and an output $[V', j]$ with $V \le V' \le V + 16$; the excess $V' - V$ is the number of duration bins of a token-and-duration transducer [28]. A diffusion denoiser has a timestep MLP from a sinusoidal width (128, 256 or 320) to a width t that feeds the modulation matrices in its blocks. A group of square maps $[C, C]$ with an aggregation matrix $[C, rC]$ is a message-passing layer with r reducers. The backbone is the largest component that owns a token table or output head, and the family (decoder, prefix LM, encoder–decoder, transducer, diffusion denoiser, encoder) follows from the components and edges.

3.5 Configuration fusion and consistency checking

When a configuration is available, the fields of Section 3.2 are read from it. Configurations of composite models nest several sub-configurations, so we first decide which one describes which component. Every object in $\mathscr{C}$ that declares a width (`hidden_size`, `d_model`, `n_embd`, `dim` and similar) is a candidate. A component is matched to the candidate whose width equals its residual width, preferring one whose layer count equals the component's and whose path names the right kind of component (`text`, `llm`, `vision`, `audio`). From the match we read the head count, the key–value head count, the head size and the routing top-k.

The match is then checked against the weights: the head count times the head size must equal the width x of the attention projections, and the key–value product must equal x_{kv}. A field that passes is shown as read from the configuration. A mismatch is reported rather than resolved, because it usually means the field has a different meaning in that family (Section 6.4). Without a configuration, the head size falls back to 128 for $d \ge 2048$ and 64 otherwise, and is shown as an estimate.

Parameters and stored bytes are sums over logical and stored tensors. KV-cache bytes per token are $2b \sum_{\ell \in \text{attn}} x_{kv}^{(\ell)}$ for b bytes per element; SSM layers keep a constant state instead. With a top-k value, active parameters per token are the non-expert parameters plus k/E of the experts. The whole procedure makes a few passes over the tree and runs in time close to linear in the number of tensors.

4 The explorer

The algorithm is the core of an open web tool [26]. We first built it for a workshop on edge inference, where participants had to understand a speech model before compressing it, and later generalized it to any architecture. Three goals guided the interface. The first screen should answer what the model is made of and how large each part is. Each value should say where it comes from, so that a reader can tell a measurement from a configuration value or an estimate. And a newcomer should be able to walk through the model from input to output, seeing what each part receives and produces.

The entry screen (Figure 2) offers three ways in and a library of analysed models. The anatomy view (Figure 3) draws the components in data-path order. Each box gives the component's role, parameters, stored size, width and depth, and a strip with one cell per layer coloured by block type, so a hybrid backbone can be read at a glance. Selecting a component lists its block types with the share of parameters per region and the

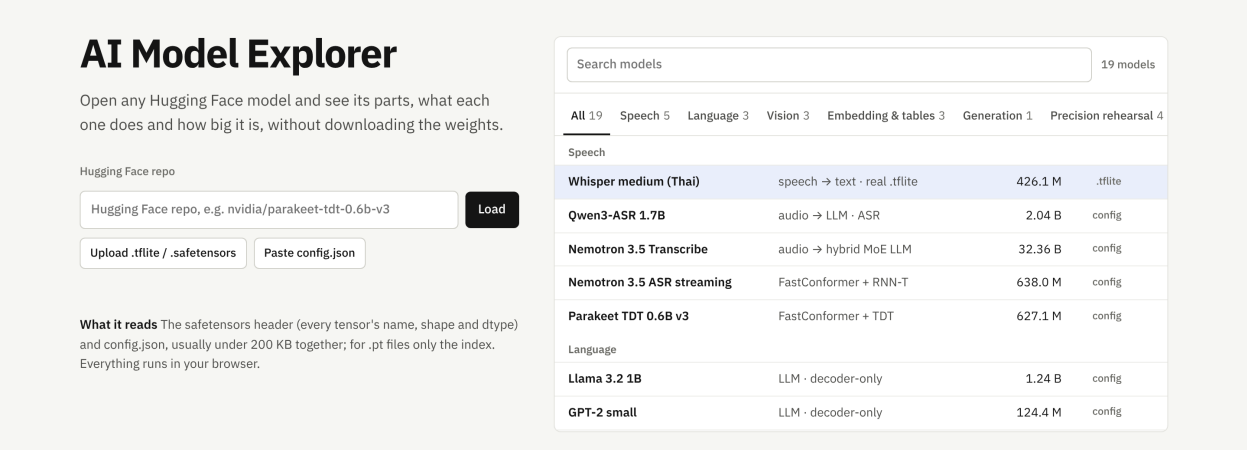


Figure 2: Entry screen. A repository id, an uploaded `.safetensors` or `.tflite` file, or a pasted configuration starts an analysis. The library on the right holds analysed models grouped by task.

shape evidence behind each label. A side panel lists the values read from the model, each tagged as measured, read from the configuration or estimated, and states whether the configuration agrees with the weights.

The big-picture view (Figure 4) lays the model out in columns from input to output. Pressing Play steps through the columns. For each one, a panel shows what goes in and what comes out, with small stand-ins for the data (a waveform, a spectrogram, a row of tokens, a matrix), the tensor shapes, and one sentence on why the part matters for optimization. The loop at the end follows the family: next token for decoders, next frame for transducers, next denoising step for diffusion models. A layer map draws every layer at its true size under switchable lenses (size, bytes read per sentence, precision), and a report lists where parameters, compute and memory traffic go, together with short observations such as "experts hold 92.8% of the weights; about 13.6 B are active per token". The interface is available in Thai and English.

As an example of use, suppose an engineer wants to run `nvidia/parakeet-tdt-0.6b-v3` on a phone. They paste the repository id and press Load. The explorer lists the repository, fetches `config.json` and the 63 KB header of the 2.5 GB weight file, and finishes in well under a second on an ordinary connection. The anatomy view shows three components: a 24-block conformer encoder with 97% of the parameters, a 12 M prediction network with a two-layer LSTM, and a 6 M joint network whose output size, 8198, is the label vocabulary plus five duration bins. The block view shows that feed-forward modules hold two thirds of each conformer block, which is where pruning or low-bit storage pays off. The play-through then makes clear that the encoder runs once per utterance while the prediction and joint networks run once per emitted label, so streaming latency depends on the small networks rather than on the encoder.

The tool is a single static page with no server, about 3,000 lines of JavaScript. The recovery code is a separate module shared with the evaluation harness, so the numbers in Section 6 measure exactly the code that users run.

5 Evaluation setup

We ask five questions. How accurately are per-tensor roles recovered, and how much depends on names (RQ1)? Are stacks, block counts and block types recovered (RQ2)? Does quantized storage change the results (RQ3)? Which hyperparameters can shapes identify, how much does the configuration add, and do configurations agree with their weights (RQ4)? What does recovery cost (RQ5)? Finally, does the method hold up on real checkpoints as they are published (RQ6)?

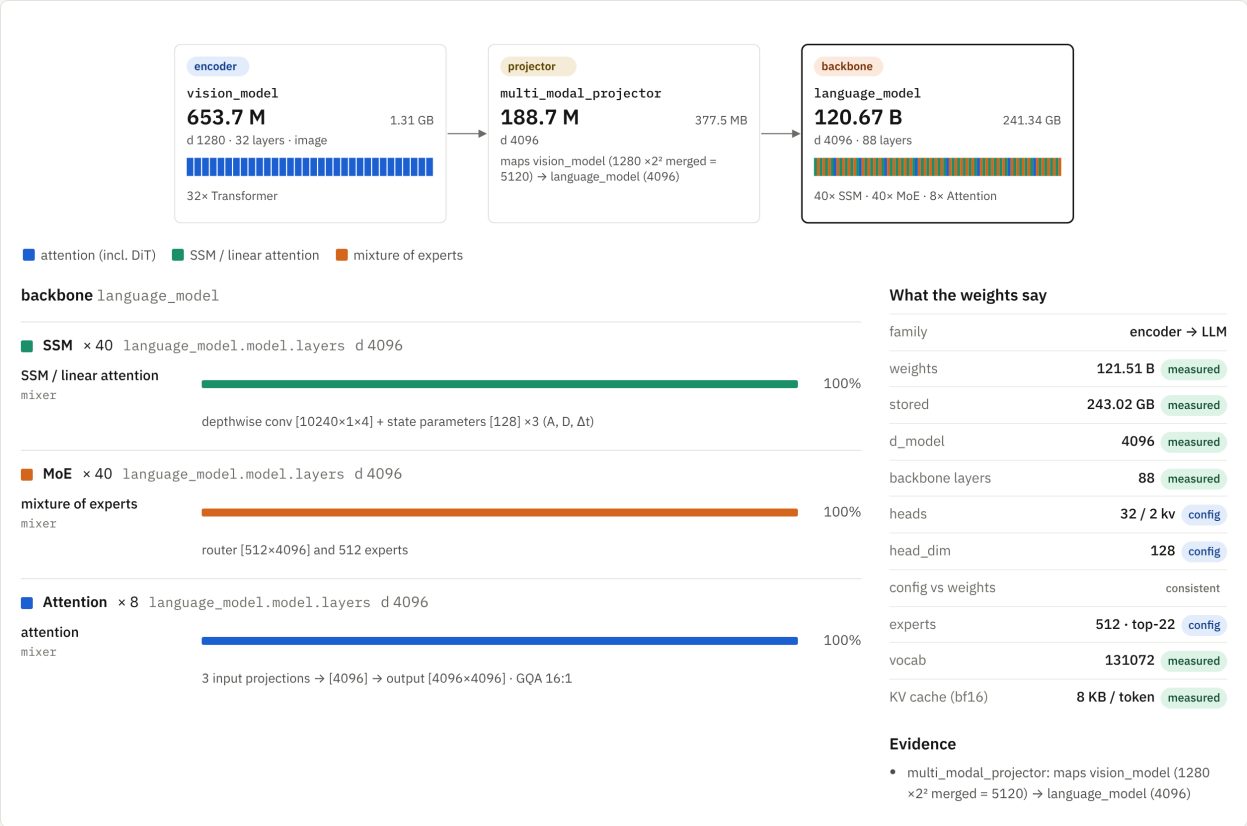


Figure 3: Anatomy view of Nemotron 3.5 Super VL (121.5 B). The three components appear in data-path order, and the backbone's strip shows how SSM, MoE and attention layers interleave. Below are the block types with the share of parameters per region and the evidence for each label; on the right, the values read from the model with their provenance and the result of the configuration check.

Corpus. The `transformers` library, version 5.22.1 [27], registers 788 configuration types. For each, we instantiate the default configuration with the most specific model class available, on PyTorch's meta device, which creates shapes without allocating memory. 548 types instantiate, and removing identical manifests leaves 510. Tied parameters are counted once and persistent buffers are kept, as in a saved checkpoint. The corpus has 268,040 tensors, a median of 419 per model, and spans 1.5 K to 1.74 T parameters with a median of 473 M. Of the 510 models, 475 contain attention, 73 mixtures of experts, 27 SSM or linear-attention mixers, 140 convolutional parts and 70 cross-attention; 231 have more than one stack and 153 have a stack with more than one block type. The configuration of each model is kept for RQ4.

Ground truth. Labels come from information the algorithm never sees: the Python class of each module. A tensor's role is given by its nearest ancestor whose class name ends in a recognized suffix (`Attention`, `MLP`, `SparseMoeBlock`, `Mamba2Mixer`, `RMSNorm` and so on), with MoE containers overriding their children and a short list of module-name fallbacks for loose tensors. Tensors that match nothing are labelled *other* and excluded. Convolutional stems and position tables outside block stacks are merged into one input-stem role, because the library is inconsistent about which class owns them. Block lists are module lists whose children each hold at least two modules with parameters, excluding expert containers and using the same sub-layer convention as the algorithm. Block types are the distinct class compositions of the items. Hyperparameters come from the configuration, using the text sub-configuration for multimodal models.

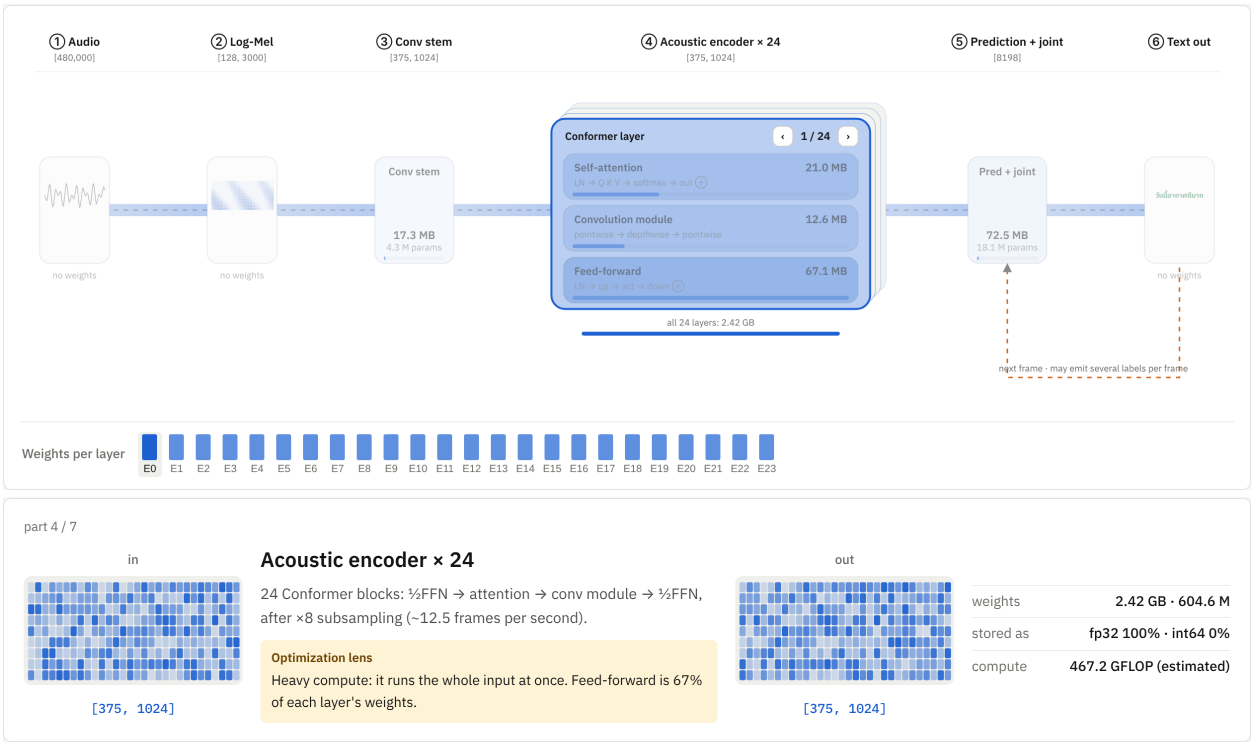


Figure 4: Big picture and play-through for Parakeet TDT 0.6B v3. Top: the transducer from audio to text, with the per-frame loop drawn back to the prediction and joint networks. Bottom: step 4 of 7, the acoustic encoder, with its input and output shapes, size, precision and estimated compute.

Metrics and conditions. Per-tensor role accuracy is weighted by parameter count within each model and then averaged over models (macro, so that each architecture counts once) or pooled over all tensors (micro, which the largest models dominate). We also report per-role F1 over tensors, recall and precision of block lists by path, the share of matched lists with the exact block count and the exact type partition, exact-match accuracy of hyperparameters, and exactness of the parameter total. The full method uses names and shapes. The shapes-only condition replaces every non-numeric path token by a consistent random hash and switches the lexicon off, which removes all lexical evidence but keeps the tree; it approximates checkpoints exported by unfamiliar code. The quantized conditions rewrite every 2-D linear weight as NVFP4, FP8 with per-channel scales, or GPTQ 4-bit (Appendix D). We compare against three baselines, all of which read tensor names, as existing tools do. The first is the set of name rules from the previous version of our explorer: regular expressions for layer indices and for fragments such as `q_proj`, `query`, `c_attn`, `mlp`, `fc1`, `encoder_attn` and `mixer`. Its label set has no mixture-of-experts role, so expert tensors fall under feed-forward; the second baseline is the same rules with expert, router and shared-expert names labelled MoE, which removes that handicap. The third is the tensor-name table that llama.cpp uses to convert Hugging Face checkpoints [7], the largest maintained mapping of module paths to tensor roles we know of (1,566 paths across its supported architectures, read from `gguf-py/gguf/tensor_mapping.py`). We map each llama.cpp tensor type to our roles, look names up after replacing layer indices, take the union over architectures because the baseline is not told the architecture, and strip expert indices as llama.cpp's converter does. For stacks, all baselines use the usual layer-index heuristic: every `<prefix>.layers.<i>` (also `layer`, `h`, `blocks`, `block`, `resblocks`) is one stack of identical blocks.

	Role accuracy (%)		Block lists (%)				Params
Condition	macro	micro	recall	precision	count	types	exact
Names and shapes	**96.8**	**99.9**	98.6	91.4	100	92.5	510/510
Shapes only	95.0	99.8	98.6	91.4	100	92.5	510/510
Name rules (v1)	69.3	8.4	60.7	72.3	100	83.4	–
Name rules (v1) + MoE label	82.3	98.4	60.7	72.3	100	83.4	–
llama.cpp name table	29.7	51.5	60.7	72.3	100	83.4	–
NVFP4 storage	96.8	99.9	98.6	91.4	100	92.5	510/510
FP8 storage	96.8	99.9	98.6	91.4	100	92.5	510/510
GPTQ storage	96.8	99.9	98.6	91.4	100	92.5	510/510
Shapes only, NVFP4 storage	95.0	99.8	98.6	91.4	100	92.5	510/510

Table 2: Main results on 510 architectures. *Count* is the share of matched lists with the exact number of blocks and *types* the share whose block-type partition equals the ground truth. The baselines share the layer-index stack heuristic, which assumes one block type per list. The micro accuracy of the v1 rules is low because the three largest models, trillion-parameter mixtures of experts, dominate the pooled score and the rules have no MoE label; with that label it rises to 98.4%.

6 Results

6.1 RQ1: Roles

Per-tensor role accuracy is 96.8% macro (Table 2). The distribution over models is skewed: the median model is labelled perfectly, 415 of 510 models reach 99% and 473 reach 90%, while 10 fall below 50%. The strongest name baseline, the v1 rules with a MoE label, reaches 82.3%. The llama.cpp table reaches 29.7% overall because it matches only exact module paths: it labels at least 80% of the tensors in 108 of the 510 models, mostly decoder language models. On those 108 it is accurate (95.9%), but our method is still higher on the same models (99.9%). Without names, all three baselines label nothing, whereas our method loses 1.8 points and 374 models still reach 99%. Most of the work is therefore done by structure and shapes rather than vocabulary.

Figure 5 breaks the result down by role. Attention, feed-forward, mixtures of experts and norms exceed 96 F1 with names and stay above 91 without them. Recurrent cells reach 100 with the lexicon but only 35 without it, because an LSTM whose input and hidden widths are equal has exactly the shapes of a feed-forward block that stores a second, integer copy of its weights, as I-BERT does. Cross-attention falls from 98 to 71 without names for a similar reason: when encoder and decoder share a width, the two attention regions of a decoder block have identical shapes, and only a name says which one reads the encoder. Output heads (54) and adapters (69) score lowest. Most of the remaining errors come from poolers and prediction transforms that the class names call heads but that look like a feed-forward or an embedding, and from single-matrix projectors inside the language model's folder.

Error analysis guided four changes made during this study, each checked on the whole corpus. Requiring the usual `ih`/`hh` naming before accepting a recurrent cell, when the lexicon is on, removed the I-BERT false positives and raised recurrent-cell F1 from 25 to 100. Treating three equal-width projections next to a convolution as attention fixed ConvBERT's span-based dynamic convolution, and excluding pointwise convolutions from the state-tensor test fixed conformer convolution modules that had been read as SSMs; together these raised convolution F1 from 77 to 83. Reading non-residual lists as heads or projectors raised head F1 from 39 to 54 and adapter F1 from 60 to 69. A rule that would have read every short convolution over the query, key and value channels as linear attention was tried and dropped: it fixed GLM-style gated delta layers but broke softmax attention layers that also use short convolutions.

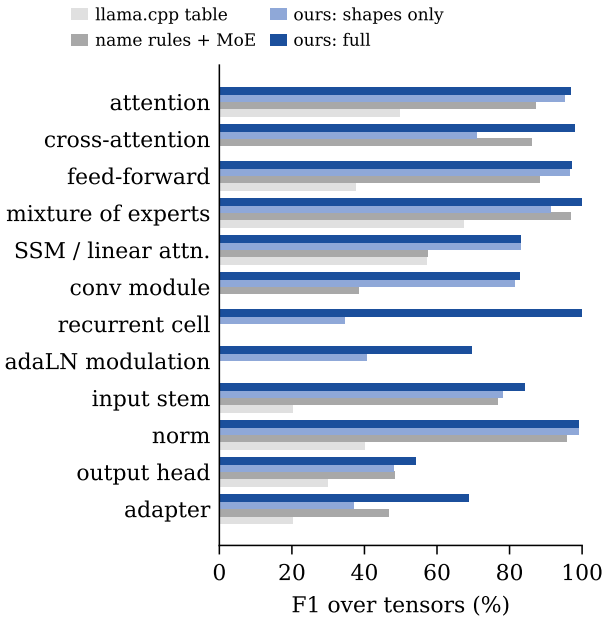


Figure 5: Per-role F1 over tensors on the corpus. The two grey bars are the name-reading baselines.

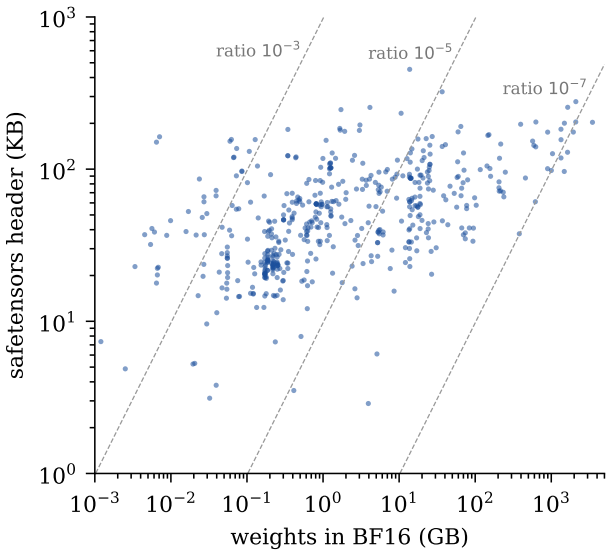


Figure 6: Header size against weight size for the 510 manifests. Dashed lines mark constant ratios. The median header is 46 KB and the largest 453 KB.

The ten models below 50% each depart from the block conventions the rules assume rather than failing on a common architecture: ALBERT shares one layer across depth and stores no list at all, PoolFormer and PatchTSMixer have blocks without attention or feed-forward signatures, and xLSTM, XLNet, MobileBERT and MobileViTv2 use block layouts that no rule describes. The linear-attention layers of GLM-5-Next keep ordinary query, key and value projections and move the decay parameters into a sub-module, so the shape rules see attention; in EdgeTAM's perceiver the opposite happens. For some modules the ground truth itself is debatable. Kimi Delta Attention and the Bailing hybrid store the same tensors as the GLM layers but their classes end in `Attention`, so the class-name oracle calls them attention.

6.2 RQ2: Stacks and block types

98.6% of ground-truth block lists (1230 of 1248) are found at the right path, and every matched list has the right number of blocks. The layer-index heuristic finds 60.7% of them, at a precision of 72.3%: it misses lists named differently (`stages`, `conformer_layers`, `transformer_blocks`, `down_blocks`) and cannot see that a list mixes block types. Precision is lower (91.4%) because the algorithm also reports repeated sub-blocks inside convolutional backbones, such as ResNet bottleneck sub-layers and SEANet residual units, which the ground-truth convention does not count. The block-type partition is exact for 92.5% of matched lists. Most of the misses are lists whose first block differs only in a module that the class signature sees and the tensor signature does not, such as a relative-position bias owned by layer 0.

6.3 RQ3: Quantized storage

Under NVFP4, FP8 and GPTQ storage the results equal those at full precision, and parameter totals stay exact for all 510 models (Table 2). This also holds with names hashed, because packed weights and their scales are recognized by dtype and shape ratios rather than by suffixes such as `weight_scale`. The point

	d	layers	FFN	experts	vocab	heads	KV heads	head size
Manifest only, all models	92.7	91.8	90.5	97.3	87.7	73.0	69.9	66.4
number of models	467	441	336	73	359	445	226	152
Manifest only, single stack	94.7	96.3	93.5	96.4	85.6	70.4	63.9	57.1
Shapes only	92.7	91.8	90.5	94.5	85.0	73.0	69.9	66.4
With configuration fusion	–	–	–	–	–	95.7	98.2	96.7
Identifiable from shapes	yes	yes	yes	yes	yes	only with per-head parameters		

Table 3: Exact-match accuracy (%) of hyperparameters against the configuration. FFN is evaluated on dense models only.

matters in practice. Hub pages count packed elements as parameters, so a 4-bit checkpoint is often shown at a fraction of its real size (Section 7).

6.4 RQ4: Identifiability, configuration fusion and consistency

Table 3 separates the fields shapes can identify from those they cannot. Width, depth, FFN size and expert count are correct for 90 to 97% of models, and more often when a model has a single stack, because then there is no doubt about which component the configuration describes. From shapes alone, the head count and head size are right about 70% of the time. The default head size is right more often (78%) than a norm vector of plausible length (72%). In latent attention, the norm before the low-rank key–value projection has the latent width rather than the head width, which is a concrete case of the limit described in Section 3.2.

With configuration fusion, head count, key–value head count and head size are right for 95.7%, 98.2% and 96.7% of models. Since the ground truth comes from the same configuration, these numbers measure the matching step, that is, whether the right sub-configuration is attached to the backbone. The backbone found a match in 455 of 510 models, and 872 of 1106 components found one. The more informative result is the consistency check. Of the 636 matched components with attention, 605 (95.1%) have configurations whose head fields multiply out to the projection widths in the weights. The 31 disagreements are concentrated in latent-attention models (DeepSeek-V2, V3 and V4, Kimi, MiniCPM3, Mistral 4, LongCat-Flash), where `head_dim` or `num_key_value_heads` refers to the query–key or latent geometry rather than to a projection width, in Falcon's fused multi-query layout, and in SAM-style windowed vision encoders. None of these configurations is wrong; each contains a field whose meaning is family-specific. A tool that trusted the configuration blindly would report wrong KV-cache sizes for exactly these models, which is why the explorer shows the disagreement instead of resolving it.

6.5 RQ5: Cost

The median safetensors header is 46 KB, the 95th percentile 164 KB and the largest 453 KB, for models of up to 3.5 TB in BF16 (Figure 6). Recovery takes 10 ms for the median model on a single JavaScript thread, 41 ms at the 95th percentile and 98 ms at most. In the explorer, network round trips dominate: a single-file model needs four requests (listing, configuration and two range requests), and a sharded model two range requests per shard, issued in parallel.

6.6 RQ6: Popular checkpoints on the Hub

The corpus measures accuracy against independent labels, but its manifests come from one library's default configurations. To test the method on checkpoints as they are actually published, we selected 201 popular public repositories across eleven groups: dense and mixture-of-experts language models, state-space and

Repository	Storage	Ours	Hub shows	Header	Recovered structure
deepseek-ai/DeepSeek-V3	FP8	684.5 B	684.5 B	11.3 MB	61 layers, 256 experts, + MTP layer
nvidia/DeepSeek-R1-0528-FP4	NVFP4	684.5 B	396.8 B	20.4 MB	same; the Hub counts packed bytes
moonshotai/Kimi-K2-Instruct	FP8	1.03 T	1.03 T	16.9 MB	61 layers, 384 experts
zai-org/GLM-4.5	BF16	358.3 B	358.3 B	5.3 MB	92 layers, 160 experts + MTP layer
Qwen/Qwen3-235B-A22B	BF16	235.1 B	235.1 B	4.4 MB	94 layers, 128 experts, 64/4 heads
MiniMaxAI/MiniMax-M2	FP8	228.7 B	228.7 B	12.0 MB	62 layers, 256 experts
openai/gpt-oss-120b	MXFP4	116.8 B	116.8 B	76 KB	36 layers, 128 experts, 64/8 heads
Qwen/Qwen3-Next-80B-A3B-Instruct	BF16	81.3 B	81.3 B	9.0 MB	48 layers, linear + full attn., 512 exp.
Qwen/Qwen2.5-VL-72B-Instruct	BF16	73.4 B	73.4 B	151 KB	vision 32 L $\rightarrow$ merger $\rightarrow$ 80 L
unsloth/Llama-3.3-70B-Instruct	BF16	70.6 B	70.6 B	83 KB	80 layers, 64/8 heads, FFN 28672
mistralai/Mixtral-8x7B-v0.1	BF16	46.7 B	46.7 B	125 KB	32 layers, 8 experts
unsloth/gemma-3-27b-it	BF16	27.4 B	27.4 B	163 KB	vision 27 L $\rightarrow$ projector $\rightarrow$ 62 L
nvidia/NVIDIA-Nemotron-Nano-9B-v2	BF16	8.89 B	8.89 B	37 KB	56 layers: Mamba-2, FFN, attention
unsloth/Qwen3-8B-bnb-4bit	NF4	8.19 B	8.41 B	193 KB	36 layers, shapes rebuilt

Table 4: A selection of the 191 checkpoints read from the Hub. *Ours* is the logical parameter count; *Hub shows* is the count on the model page. *Header* is the number of bytes read.

	d	layers	FFN	experts	vocab	heads	KV heads	head size
Number of models	155	154	112	31	145	149	94	56
Manifest only	95.5	92.9	90.2	100	91.7	77.2	75.5	83.9
With configuration fusion	–	–	–	–	–	96.6	97.9	96.4

Table 5: Exact-match accuracy (%) of hyperparameters on the Hub checkpoints, against each repository's `config.json`.

hybrid models, encoders and embedding models, sequence-to-sequence models, speech and audio models, vision backbones, vision–language models, image and video generators, time-series and tabular models, and quantized releases. Gated models were taken from ungated mirrors where one exists. Together the repositories hold 12.1 TB of weights and were downloaded about 505 million times in the month before the run. A benchmark page in the explorer (`bench.html`) runs the same loader as the tool in the user's browser and exports each manifest, the configuration files and the Hub's metadata; the analysis then runs offline with the same recovery code.

191 repositories were read. Of the ten failures, seven are gated, two store old non-zip pickles, and one, MMS-1B-all, ships one adapter file per language, and reading all their headers triggered the Hub's rate limit; the loader now skips such adapters. Reading the 191 manifests transferred 161 MB in total. The median repository needed 79 KB, the 95th percentile 5.4 MB, and the largest, an NVFP4 release of DeepSeek-R1 with 163 shards and 180,488 tensors, 20.8 MB. Recovery took 18 ms for the median model and 4.2 s for the largest.

Parameter counts. The Hub's model pages show a parameter count computed from safetensors metadata, which gives an independent check of the header reading. Excluding 0-d scalars, which the Hub skips, our count equals the Hub's for 151 of the 158 transformers-style repositories that have one: the stored count for full-precision files, and the logical count for FP8, GPTQ, AWQ, MXFP4 and MLX releases, for which the Hub reports logical sizes. For the 12 diffusers pipelines with a count, the Hub's figure equals exactly the parameters we assign to the denoiser (the UNet or transformer folder), while our total also includes the text encoders and VAE. Of the seven remaining differences, four are 4-bit releases where the Hub counts packed storage: an NVFP4 DeepSeek-R1 is shown at 396.8 B instead of 684.5 B (Table 4), NVFP4 8 B models are shown at about 57% of their size, and bitsandbytes releases 2 to 3% above it, because packed bytes, block scales and quantization maps are counted as parameters. The other three come from extra files

Group	Repos	Read	Hub =	Heads (fused)	Roles	Median header	Median time
Dense LM	48	45	44/44	45/45	99.8% (40)	38 KB	7 ms
MoE LM	24	24	24/24	24/24	100.0% (3)	2.2 MB	630 ms
SSM / hybrid	10	9	8/8	4/6	68.3% (5)	66 KB	18 ms
Encoder	8	7	6/6	6/7	92.1% (7)	23 KB	4 ms
Embedding	10	10	9/9	10/10	100.0% (1)	34 KB	8 ms
Seq2seq	7	6	3/3	6/6	100.0% (6)	155 KB	11 ms
Speech / audio	25	23	15/16	17/17	99.7% (11)	123 KB	13 ms
Vision	19	19	16/16	7/9	94.7% (6)	62 KB	10 ms
Vision–language	19	18	15/17	12/12	–	107 KB	27 ms
Image / video gen.	15	14	0/0	–	–	338 KB	46 ms
Time series / tabular	6	6	5/5	3/3	97.7% (2)	31 KB	5 ms
Quantized release	10	10	6/10	10/10	100.0% (1)	109 KB	13 ms

Table 6: Hub benchmark by group. *Hub* = counts transformers-style repositories whose total agrees with the model page; diffusers pipelines are excluded here (their Hub count equals our denoiser in all 12 cases). *Heads (fused)* is the head count after configuration fusion, against `config.json`; *Roles* is macro role accuracy on models whose type is in the corpus (number of models in brackets).

that the Hub does not count, such as a codec stored beside a speech model and LoRA folders shipped with Phi-4-multimodal.

Hyperparameters and roles. Table 5 repeats the analysis of RQ4 on the Hub checkpoints. The pattern is the same as on the corpus, with higher numbers because popular checkpoints are mostly single-stack models with standard layouts. From the manifest alone, the head count is right for 77% of models; with fusion it is right for 97%, and the configuration agrees with the weights for 140 of the 143 checked models. For 82 checkpoints whose model type is also in the corpus, we transferred the class-name labels by index-masked tensor name. Role accuracy on these is 96.8% macro, and 71 of the 82 reach 99%, against 91.9% for the v1 rules with a MoE label and 63.2% for the llama.cpp table. On the 51 of these checkpoints that the llama.cpp table covers well (at least 80% of tensors labelled), it reaches 97.7% and our method 99.7%. For the layer count, the layer-index heuristic matches the configuration for 138 of 154 models and our method for 143. The four below 90% are xLSTM, ALBERT and Zamba2, which also fail on the corpus, and RT-DETR, whose CNN backbone the rules split into many small stacks. Table 6 breaks the results down by group, and Appendix F lists every checkpoint.

What real checkpoints added. Four problems appeared only on real files, and each led to a change that was then checked on the corpus. DeepSeek-V3 and GLM-4.5 store their multi-token-prediction layer as one extra item of the main layer list. It is now recognized by its $[d, 2d]$ projection and two extra norms and reported separately, so the layer count matches the configuration (61 and 92 rather than 62 and 93). bitsandbytes releases keep the logical shape in a serialized state blob that the header does not expose; the loader now regroups the state tensors with their weight, counts two values per packed byte, and rebuilds a matrix shape from the residual width, which recovers the 8.19 B of Qwen3-8B. In Qwen3-0.6B the key and value projections are square $[d, d]$, which had made one of them look like the output projection; the output is now taken from the non-square side. Whisper's encoder stores a position table with 1,500 rows that had been read as a token table, which made the encoder look like the backbone. A component fed by a convolutional stem is no longer treated as having a token table. On the loading side, repositories that ship several copies of the same weights (FP16 and FP32 files, single-file checkpoints beside a diffusers layout, LoRA files) are now reduced to one description, because the first run read some of them more than once.

Model	Family	Params	Header	Recovered structure
Nemotron 3.5 Super VL	prefix LM	121.5 B	130 KB	ViT 32 L → projector → 40 SSM, 40 MoE, 8 attn.
Nemotron 3.5 Transcribe	prefix LM	32.4 B	154 KB	3 encoders → projector → 23 SSM, 23 MoE, 6 attn.
Nemotron 3.5 ASR streaming	transducer	638 M	57 KB	24 conformer blocks, LSTM, RNN-T joint
Parakeet TDT 0.6B v3	transducer	627 M	63 KB	same encoder, TDT joint, 5 duration bins
Nemotron 3 Embed 1B NVFP4	encoder	1.14 B	41 KB	16 blocks, 112 NVFP4 linear layers
FastH3 Omni FP8†	diffusion	33.1 B	84 KB	2 refiner + 50 DiT blocks, 300 FP8 tensors
Kumo Tabular (large)†	encoder	214 M	41 KB	row embedding → 24-layer backbone
Kumo Relational†	encoder	29 M	24 KB	rows → message passing → 12 layers

Table 7: Anatomies recovered for models released in 2026. Manifests marked † were rebuilt from public source code and configuration because `transformers` has no reference implementation for them.

7 Case studies

Table 7 lists eight models released in 2026 that we studied in detail. They were not in the corpus, but they were not held out from development either: we studied them while extending the rules, and several parts of Section 3.4, including transducer joints, timestep embedders, message-passing layers and adapter components, were written because of them. Every change was checked on the full corpus for regressions. The case studies therefore show that the representation is rich enough for these architectures, while the unbiased accuracy estimate is the one in Section 6.

The Super VL backbone mixes three block types in one list, and the recovered type sequence reproduces the configuration's `layers_block_type` exactly. The projector's input width, 5120, is four times the vision width, and the algorithm reports a 2×2 merge, matching the configuration's `downsample_ratio` of 0.5. Experts hold 112.7 B of the 121.5 B parameters. With the configured top-22 routing this gives 13.6 B active parameters per token, including the vision tower and both embedding tables, against the 12 B in the model's name. Only 8 of the 88 layers keep a KV cache, each with two key–value heads, so the cache costs 8 KB per token in BF16, an eleventh of what it would be if every layer used the same attention.

For both speech transducers the joint network is found from three shapes, and the number of duration bins falls out as a difference. Parakeet's joint output has 8198 rows and its label table 8193, giving the five durations $\{0, \ldots, 4\}$ of its configuration, while the streaming model's two sizes are equal, as in a standard RNN-T. Parakeet's parameter total differs from the Hub's figure by 24, the number of batch-norm step counters, one per conformer block. The NVFP4 embedding model has 1.14 B logical parameters, matching its model card, while the Hub reports 0.7 B because it counts packed `U8` elements. That it is an encoder cannot be seen in the weights; its configuration's `is_causal: false` settles it, and the explorer shows the family as read from the configuration.

Besides the new components, four fixes to existing rules came from these models: the residual width of a block is taken from its norms, `nn.Sequential` children are no longer split as sub-layers, a module that wraps two different sub-blocks is split, and in repositories with several top-level folders a component is never raised above its folder. One part stays unexplained: the streaming model's language-prompt projector remains unattached, because its prompt table lives outside the components the manifest reveals.

8 Discussion and limitations

Class names are an independent source of labels but not a perfect one. Some implementations place a position table inside a class called `...Embeddings` and others inside the encoder; some call a pooler a `...Head`; and linear-attention modules are named `...Attention` in some families and `...GatedDeltaNet` in others. We merged roles where the library is inconsistent and kept the fallback list short, but per-role numbers

for heads and adapters should be read with this in mind. The headline numbers are robust to it, because these roles carry little of the parameter mass.

All corpus manifests come from `transformers` implementations, which share naming habits. The shapes-only condition is the main protection against overfitting to those habits, and its 1.8-point drop suggests that the structural rules transfer. Default configurations are not trained checkpoints, but their shapes, which are all the method reads, follow the same rules. The checkpoint benchmark of Section 6.6 addresses this concern directly, with real files from many organizations. Fusion accuracy is measured against the same configuration that supplies the fields, so it validates the matching step and not the configuration; the consistency check is the independent part, and a richer check could model latent attention explicitly.

GGUF and ONNX files are not handled yet, although both carry comparable metadata, and old non-zip PyTorch pickles are rejected. Order inside a block and activation functions remain unknown, and compute estimates assume standard computation per role. The method reads only what a hub already serves publicly. It never downloads or executes weights, and the PyTorch reader interprets a fixed subset of pickle opcodes as data without resolving Python callables.

9 Conclusion

A tensor manifest is small, safe and almost always available, and it carries more of an architecture than one might expect. With the module tree, a few dozen shape rules and width matching, we recover blocks, block types, regions, data paths and parameter counts for 510 architectures at 96.8% role accuracy, keep most of that accuracy without any names, and stay exact on parameter counts under current quantized formats. The same analysis makes the limits explicit: head geometry, masking and routing are not in the shapes. Taking them from the configuration closes most of the gap, and checking the configuration against the weights exposes the families whose fields mean something else. We hope the benchmark and the explorer make it routine to look inside a model before downloading it.

Availability. The explorer runs at `https://joessattes.github.io/ai-model-explorer/` and the checkpoint benchmark at `https://joessattes.github.io/ai-model-explorer/bench.html`. Source code, the corpus builder, the evaluation scripts and the sources of this paper are at `https://github.com/JoesSattes/ai-model-explorer` under the MIT license. The corpus results are reproduced by `python research/build_corpus.py corpus.jsonl` followed by `node research/eval.js corpus.jsonl` with the flags `-obf -nolex`, `-quant nvfp4`, `-v1` and `-cfg`; the checkpoint results by `node research/realbench.js export.jsonl corpus.jsonl`.

References

[1] Joshua Ainslie, James Lee-Thorp, Michiel de Jong, Yury Zemlyanskiy, Federico Lebrón, and Sumit Sanghai. GQA: Training generalized multi-query transformer models from multi-head checkpoints. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, 2023.

[2] Aeree Cho, Grace C. Kim, Alexander Karpekov, Alec Helbling, Zijie J. Wang, Seongmin Lee, Benjamin Hoover, and Duen Horng Chau. Transformer explainer: Interactive learning of text-generative models. *arXiv preprint arXiv:2408.04619*, 2024.

[3] Tri Dao and Albert Gu. Transformers are SSMs: Generalized models and efficient algorithms through structured state space duality. In *International Conference on Machine Learning*, 2024.

[4] DeepSeek-AI. DeepSeek-V2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[5] William Fedus, Barret Zoph, and Noam Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[6] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. GPTQ: Accurate post-training quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023.

[7] Georgi Gerganov and ggml-org contributors. llama.cpp: LLM inference in C/C++. https://github.com/ggml-org/llama.cpp, 2023. gguf-py/gguf/tensor_mapping.py, read 1 November 2026.

[8] Google AI Edge. Model explorer: Intuitive and hierarchical visualization of model graphs. https://ai.google.dev/edge/model-explorer, 2024.

[9] Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.

[10] Albert Gu and Tri Dao. Mamba: Linear-time sequence modeling with selective state spaces. *arXiv preprint arXiv:2312.00752*, 2023.

[11] Anmol Gulati, James Qin, Chung-Cheng Chiu, Niki Parmar, Yu Zhang, Jiahui Yu, Wei Han, Shibo Wang, Zhengdong Zhang, Yonghui Wu, and Ruoming Pang. Conformer: Convolution-augmented transformer for speech recognition. In *Interspeech*, 2020.

[12] Noah Hollmann, Samuel Müller, Lennart Purucker, Arjun Krishnakumar, Max Körfer, Shi Bin Hoo, Robin Tibor Schirrmeister, and Frank Hutter. Accurate predictions on small data with a tabular foundation model. *Nature*, 637:319–326, 2025.

[13] Xing Hu, Ling Liang, Shuangchen Li, Lei Deng, Pengfei Zuo, Yu Ji, Xinfeng Xie, Yufei Ding, Chang Liu, Timothy Sherwood, and Yuan Xie. Deepsniffer: A dnn model extraction framework based on learning architectural hints. In *Proceedings of the 25th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, pages 385–399, 2020.

[14] Hugging Face. safetensors: Simple, safe way to store and distribute tensors. https://github.com/huggingface/safetensors, 2022.

[15] Ji Lin, Jiaming Tang, Haotian Tang, Shang Yang, Wei-Ming Chen, Wei-Chen Wang, Guangxuan Xiao, Xingyu Dang, Chuang Gan, and Song Han. AWQ: Activation-aware weight quantization for on-device LLM compression and acceleration. In *Proceedings of Machine Learning and Systems*, 2024.

[16] Paulius Micikevicius, Dusan Stosic, Neil Burgess, Marius Cornea, Pradeep Dubey, Richard Grisenthwaite, Sangwon Ha, Alexander Heinecke, Patrick Judd, John Kamalu, et al. FP8 formats for deep learning. *arXiv preprint arXiv:2209.05433*, 2022.

[17] Margaret Mitchell, Simone Wu, Andrew Zaldivar, Parker Barnes, Lucy Vasserman, Ben Hutchinson, Elena Spitzer, Inioluwa Deborah Raji, and Timnit Gebru. Model cards for model reporting. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 220–229, 2019.

[18] NVIDIA. Nemotron-H: A family of accurate and efficient hybrid Mamba-Transformer models. *arXiv preprint arXiv:2504.03624*, 2025.

[19] NVIDIA. Pretraining large language models with NVFP4. *arXiv preprint arXiv:2509.25149*, 2025.

[20] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

[21] Jingang Qu, David Holzmüller, Gaël Varoquaux, and Marine Le Morvan. TabICL: A tabular foundation model for in-context learning on large data. In *Proceedings of the 42nd International Conference on Machine Learning*, pages 50817–50847, 2025.

[22] Dima Rekesh, Nithin Rao Koluguri, Samuel Kriman, Somshubra Majumdar, Vahid Noroozi, He Huang, Oleksii Hrinchuk, Krishna Puvvada, Ankur Kumar, Jagadeesh Balam, and Boris Ginsburg. Fast conformer with linearly scalable attention for efficient speech recognition. In *IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, 2023.

[23] Lutz Roeder. Netron: Visualizer for neural network, deep learning and machine learning models. https://github.com/lutzroeder/netron, 2017.

[24] Bita Darvish Rouhani, Ritchie Zhao, Ankit More, Mathew Hall, Alireza Khodamoradi, Summer Deng, Dhruv Choudhary, Marius Cornea, Eric Dellinger, Kristof Denolf, et al. Microscaling data formats for deep learning. *arXiv preprint arXiv:2310.10537*, 2023.

[25] Noam Shazeer, Azalia Mirhoseini, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, and Jeff Dean. Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*, 2017.

[26] Sattaya Singkul. AI Model Explorer. https://joessattes.github.io/ai-model-explorer/, 2026.

[27] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, 2020.

[28] Hainan Xu, Fei Jia, Somshubra Majumdar, He Huang, Shinji Watanabe, and Boris Ginsburg. Efficient sequence transduction by jointly predicting tokens and durations. In *International Conference on Machine Learning*, 2023.

[29] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium*, pages 2003–2020, 2020.

[30] Songlin Yang, Jan Kautz, and Ali Hatamizadeh. Gated delta networks: Improving Mamba2 with delta rule. In *International Conference on Learning Representations*, 2025.

[31] Boyi Zeng, Lizheng Wang, Yuncong Hu, Yi Xu, Chenghu Zhou, Xinbing Wang, Yu Yu, and Zhouhan Lin. HuRef: HUman-REadable fingerprint for large language models. In *Advances in Neural Information Processing Systems*, 2024.

[32] Jian Zhao, Shenao Wang, Yanjie Zhao, Xinyi Hou, Kailong Wang, Peiming Gao, Yuanchao Zhang, Chen Wei, and Haoyu Wang. Models are codes: Towards measuring malicious code poisoning attacks on pre-trained model hubs. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2024.

A Corpus construction

The model class for each configuration type is chosen in this order: image-text-to-text, multimodal LM, speech sequence-to-sequence, sequence-to-sequence LM, TDT, RNN-T, causal LM, CTC, masked LM, image classification, text-to-waveform, object detection, base model. Instantiation uses `_from_config` under `torch.device("meta")` with a 90-second limit. Of the 240 types that do not instantiate, 177 are sub-configurations without a model class of their own, such as vision or audio towers registered separately, and the rest fail on required sub-configurations or optional dependencies. Manifests are deduplicated by the exact list of names and shapes, and tied parameters are kept once under the name PyTorch reports first.

Each record stores the manifest (name, dtype and shape of every tensor), the per-tensor ground-truth role, the module lists with their item classes, the parameter total, and the serialized configuration. Table 8 gives the families our method assigns, which approximates the composition of the corpus; "encoder" covers bidirectional encoders, vision backbones and decoders with tied embeddings, which are indistinguishable from the manifest alone (Section 3.2).

	encoder	decoder	prefix LM	enc–dec	transducer	diffusion	unknown
Models	356	93	46	8	3	2	2

Table 8: Families assigned to the 510 corpus models.

B Ground-truth labelling

A tensor's label comes from the classes of the modules on its path. If any ancestor class matches the MoE pattern, the label is mixture of experts. Otherwise the nearest ancestor whose lower-cased class name matches one of the patterns in Table 9, tried in the order shown, gives the label; an attention match whose module path contains `encoder_attn` or `cross_attn` becomes cross-attention. Tensors that no class matches fall back to their module name: `lm_head`, `classifier`, `score` or `joint` give head; `fc`*n*, `mlp`, `dense`, `w`*n*, `c_fc` and `c_proj` give feed-forward; `conv`, `patch` or `subsampl` outside the layer lists give input stem; `merger`, `projector` or `connector` give adapter. Everything else is *other* and is excluded from scoring.

C Lexicon and thresholds

The optional lexicon consists of eight name patterns, one per region role, plus patterns for the query, key, value, output, gate, up and down projections inside a region. It is consulted in four places only: to pick between two region labels whose shape signatures are equally strong; to tell a token table from an untied output head when both have shape $[V, d]$ and neither position nor nesting decides; to accept a recurrent cell, which additionally requires `ih`/`hh` or a cell name; and to name loose projectors whose shapes match two components. In the shapes-only condition all four fall back to shape rules or to the label *other*.

The numeric thresholds are the following. Depthwise convolutions count as SSM convolutions up to kernel size 8 and as conformer convolutions above it or when pointwise convolutions are present. Per-head gates have at most 64 rows. A fused feed-forward requires $f > 1.3d$. A router may have up to eight more rows than there are experts. Cross-attention requires two projections from a foreign width of at least 128. adaLN requires a conditioning width of at least 32. Adapters allow pixel-shuffle factors $k \in \{2, 3, 4\}$. A transducer joint may have up to 16 outputs more than the label table. Without a configuration, the head size is

Label	Class-name pattern (suffix of the lower-cased class name)
MoE (any ancestor)	`sparsemoeblock`, `moeblock`, `sparsemoe`, `moe`, `moelayer`, `experts`, `expert`, `router`, `topkrouter`, `moegate`, `sparsemlp`, `routedexperts`
head	`predictionheadtransform`, `lmpredictionhead`, `predictionhead`, `lmhead`, `pooler`, `classificationhead`, `mlmhead`, `predictions`
attention	`selfoutput`, `attentionoutput`
norm	`rmsnorm`, `layernorm`, `groupnorm`, `batchnorm1d/2d`, `norm`
cross-attention	`crossattention`, `encdecattention`, `crossattn`
SSM / linear attn.	`mamba(2)mixer`, `mixer`, `mamba`, `ssm`, `deltanet`, `gateddeltanet`, `linearattention`, `lightningattention`, `rwkv*`, `recurrentblock`, `rglru`, `xlstm*`, `retention`, `hgrn`, `gla`
attention	`attention`, `attn`, `selfattention`, `multiheadattention`, `sdpaattention`
recurrent cell	`lstm`, `gru`, `rnn`, `lstmcell`
adaLN modulation	`adalayernorm*`, `adaln*`, `modulation`, `timestepembedding`, `timestepembedder`
convolution module	`convolutionmodule`, `convmodule`, `conformerconvolution*`, `convblock`, `depthwiseconv*`
feed-forward	`mlp`, `feedforward`, `ffn`, `intermediate`, `output`, `swiglu`, `geglu`, `densereludense`, `densegatedactdense`
input stem	`embedding(s)`, `embed`, `patchembed*`

Table 9: Class-name patterns used for the ground truth, in the order they are tried.

128 for $d \geq 2048$ and 64 otherwise. A multi-token-prediction layer is the last item of a stack that holds a $[d, 2d]$ matrix and at least two more width-d vectors than the item before it.

D Synthetic quantized storage

The rewrite applies to every 2-D tensor named `*.weight` whose input dimension is a multiple of 32 and output dimension a multiple of 8, excluding embeddings, heads and norms. NVFP4 replaces $[o, i]$ by `U8` $[o, i/2]$, `F8_E4M3` $[o, i/16]$ and two `F32` scalars. FP8 uses `F8_E4M3` $[o, i]$ and `F32` $[o, 1]$. GPTQ uses `qweight` as `I32` $[i/8, o]$, `qzeros` $[i/128, o/8]$, `scales` as `F16` $[i/128, o]$ and `g_idx` $[i]$. Under name hashing these suffixes are hashed too, so recognition relies on dtypes and shape ratios. On real checkpoints the loader also handles MXFP4 (`_blocks` and `_scales` pairs), MLX (`U32` words with `scales` and `biases`), and bitsandbytes NF4, whose state tensors are stored under `<module>.weight.*`.

E Corpus results per model

Table 10 lists the most frequent confusions and Table 11 the role accuracy of every corpus model, with and without names.

Ground truth → predicted	Tensors	Models (examples)
attn → conv	1387	cvt, efficientloftr
ssm → attn	1130	glm5_next
ffn → other	871	albert, cvt
attn → ssm	676	edgetam_video
ffn → head	641	blt, canine, cohere2_vision
attn → norm	593	mistral_large4
attn → other	553	albert
attn → head	328	blt
ffn → conv	302	d_fine, deimv2
ffn → attn	284	evolla
attn → ffn	198	bridgetower
norm → head	158	aimv2, aimv2_vision_model
embed → attn	140	gemma3n_audio
embed → conv	129	cvt
ssm → conv	120	minimax
cross → attn	116	blt

Table 10: The most frequent confusions of the full method on the corpus, counted in tensors.

Table 11: Per-model role accuracy (%) on the corpus: full method / shapes only. Model types are the `transformers` configuration names.

Type	Full	Shapes	Type	Full	Shapes
afmoe	100.0	100.0	minimax	97.1	97.1
aimv2	100.0	100.0	minimax_m2	100.0	100.0
aimv2_vision_model	100.0	100.0	minimax_m3_vl	100.0	100.0
albert	0.0	0.0	minimax_m3_vl_text	100.0	100.0
align	100.0	100.0	ministral3	100.0	100.0
altclip	100.0	100.0	mistral	100.0	100.0
apertus	100.0	100.0	mistral3	100.0	100.0
arcee	100.0	100.0	mistral4	100.0	100.0
aria_text	100.0	100.0	mistral_large4	100.0	100.0
audio-spectrogram-transformer	100.0	100.0	mistral_large4_text	100.0	100.0
audioflamingo3	100.0	100.0	mixtral	100.0	100.0
audioflamingo3_encoder	100.0	100.0	mlcd_vision_model	100.0	100.0
axk1	100.0	100.0	mllama	100.0	91.5
axk2	100.0	100.0	mm-grounding-dino	88.5	86.0
bailing_hybrid	100.0	100.0	mobilebert	38.1	36.9
bamba	100.0	100.0	mobilenet_v1	100.0	99.9
bark	81.9	94.0	mobilenet_v2	100.0	99.9
bart	100.0	100.0	mobilevit	100.0	100.0
beam	100.0	100.0	mobilevitv2	39.8	39.8
beit	100.0	100.0	modernbert	100.0	100.0
bert	100.0	100.0	modernbert-decoder	100.0	100.0
bert-generation	100.0	100.0	modernvbert	97.6	97.6
big_bird	100.0	100.0	molmo2	98.6	91.5
bigbird_pegasus	98.5	98.5	molmo2_text	100.0	100.0
biogpt	100.0	100.0	moonshine	100.0	100.0
bit	100.0	100.0	moshi	100.0	98.1
bitnet	100.0	100.0	moss_transcribe_diarize	100.0	99.9
blenderbot	100.0	100.0	moss_transcribe_diarize_audio	100.0	100.0
blenderbot-small	99.4	99.4	mpnet	100.0	100.0
blip	100.0	100.0	mpt	100.0	100.0
blip-2	100.0	100.0	mra	100.0	100.0
blip_2_qformer	100.0	100.0	mt5	100.0	96.3
bloom	100.0	100.0	muse_glimmer	99.9	99.9
blt	98.7	98.7	muse_glimmer_assistant	100.0	100.0
bridgetower	82.6	82.6	muse_glimmer_text	100.0	100.0

Type	Full	Shapes	Type	Full	Shapes
bros	100.0	100.0	muse_glimmer_vision	99.9	99.9
camembert	100.0	78.6	muse_spark	100.0	100.0
canary	100.0	100.0	muse_spark_audio	91.7	88.3
canine	89.3	79.7	muse_spark_text	100.0	100.0
chinese_clip	100.0	100.0	mvp	100.0	100.0
chinese_clip_vision_model	100.0	100.0	nanochat	100.0	100.0
clap	100.0	100.0	nemo_speechlm	100.0	100.0
clip	100.0	100.0	nemotron3_5_asr	99.8	96.8
clip_text_model	100.0	99.9	nemotron3_5_transcribe_asr_audio	100.0	100.0
clipseg	100.0	100.0	nemotron3_5_transcribe_audio	100.0	100.0
clvp	100.0	99.9	nemotron3_5_transcribe_word_timestamp	100.0	100.0
codegen	100.0	96.6	nemotron3_diarization	100.0	100.0
cohere	100.0	100.0	nemotron3_diarization_audio	100.0	100.0
cohere2_moe	100.0	100.0	nemotron_asr_streaming	99.9	99.6
cohere2_vision	100.0	99.5	nemotron_asr_streaming_encoder	100.0	100.0
cohere_asr	99.9	99.9	nemotron_h	100.0	100.0
cohere_compass_vision	99.5	72.9	nemotron_h_omni	100.0	100.0
convbert	100.0	100.0	neomme	28.8	0.9
convnext	99.6	99.5	neucodec	100.0	100.0
convnextv2	100.0	99.9	nllb-moe	100.0	6.1
cosmos3_edge	100.0	100.0	nomic_bert	100.0	100.0
cosmos3_edge_text	100.0	100.0	nystromformer	100.0	100.0
cosmos3_edge_vision	99.8	99.8	olmo	100.0	100.0
cosmos3_omni	100.0	98.8	olmo2	100.0	100.0
cpmant	100.0	98.7	olmo_hybrid	100.0	100.0
csm	99.7	99.7	olmoe	100.0	100.0
ctrl	100.0	100.0	oneformer	99.9	85.5
cvt	95.7	95.7	openai-gpt	100.0	100.0
cwm	100.0	100.0	openai_privacy_filter	100.0	100.0
d_fine	92.0	92.0	opt	100.0	100.0
dac	99.9	99.9	ovis2	100.0	100.0
data2vec-audio	96.0	96.0	owlv2	100.0	100.0
data2vec-text	100.0	100.0	owlvit	100.0	100.0
data2vec-vision	100.0	100.0	paddleocr_vl	100.0	100.0
deberta	100.0	100.0	paligemma	100.0	100.0
deberta-v2	100.0	100.0	parakeet_ctc	100.0	100.0
decision_transformer	89.5	89.5	parakeet_encoder	100.0	100.0
deepseek_v2	100.0	100.0	parakeet_rnnt	99.8	98.2
deepseek_v3	100.0	100.0	parakeet_tdt	99.8	98.2
deepseek_v32	100.0	100.0	patchtsmixer	10.3	10.3
deepseek_v4	100.0	100.0	patchtst	100.0	100.0
deepseek_vl	99.9	99.9	pe_audio	100.0	100.0
deepseek_vl_hybrid	99.9	99.9	pe_audio_encoder	100.0	100.0
deimv2	91.4	91.4	pegasus	99.5	99.5
deit	100.0	100.0	pegasus_x	100.0	100.0
depth_pro	100.0	100.0	perceiver	96.4	95.7
dia	100.0	100.0	persimmon	100.0	100.0
diffllama	100.0	100.0	phi	100.0	100.0
dinov2	100.0	100.0	phi3	100.0	100.0
dinov2_with_registers	99.8	99.8	phi4_multimodal	99.0	99.0
dinov3_convnext	100.0	100.0	phimoe	100.0	100.0
dinov3_vit	100.0	100.0	pi0	84.9	99.9
distilbert	100.0	100.0	pix2struct	98.7	61.3
doge	100.0	100.0	pixio	100.0	100.0
donut-swin	100.0	100.0	pixtral	100.0	100.0
dpr	100.0	100.0	plbart	100.0	100.0
dpt	100.0	100.0	pocket_tts	95.3	95.3
edgetam_video	90.8	89.3	pocket_tts_audio_tokenizer	100.0	100.0
efficientloftr	78.6	78.6	pocket_tts_audio_tokenizer_decoder	100.0	100.0
efficientnet	100.0	100.0	pocket_tts_audio_tokenizer_encoder	100.0	100.0
electra	99.3	70.3	poolformer	0.1	0.1
encodec	100.0	100.0	pop2piano	100.0	89.3
encoder-decoder	90.5	81.0	pp_doclayout_v2	96.6	96.6
ernie	100.0	100.0	pp_doclayout_v3	96.5	96.5
ernie4_5	100.0	100.0	pp_doclayout_v4	96.8	96.8
ernie4_5_moe	100.0	100.0	pp_formulanet	100.0	99.1

Type	Full	Shapes	Type	Full	Shapes
ernie4_5_vl_moe	99.5	99.5	pp_lcnet	100.0	99.6
esmc	100.0	100.0	pp_ocrv5_mobile_det	100.0	100.0
eurobert	100.0	100.0	pp_ocrv5_mobile_rec	100.0	100.0
evolla	84.5	89.7	pp_ocrv5_server_rec	100.0	100.0
exaone4	100.0	100.0	pp_ocrv6_small_det	100.0	100.0
exaone4_5	99.6	97.9	pp_ocrv6_small_rec	100.0	100.0
exaone4_5_vision	98.1	90.4	prophetnet	100.0	100.0
exaone_moe	100.0	100.0	pvt	82.5	82.5
falcon	100.0	100.0	pvt_v2	53.0	53.0
falcon_h1	100.0	100.0	qianfan_ocr	100.0	99.7
falcon_mamba	100.0	75.8	qianfan_ocr_vision	100.0	100.0
fastspeech2_conformer	99.7	99.7	qwen2	100.0	100.0
fastspeech2_conformer_with_hifigan	99.9	99.9	qwen2_5_omni	99.7	98.4
flaubert	100.0	100.0	qwen2_5_omni_thinker	99.5	97.7
flava	100.0	99.5	qwen2_5_vl	99.9	99.7
flex_olmo	100.0	100.0	qwen2_5_vl_text	100.0	100.0
flint	100.0	100.0	qwen2_5_vl_vision	98.3	91.7
florence2	95.1	93.0	qwen2_audio	100.0	100.0
fnet	99.3	99.3	qwen2_moe	100.0	100.0
focalnet	86.0	62.6	qwen2_vl	100.0	99.9
fsmt	90.2	90.2	qwen3	100.0	100.0
fun_asr_nano	73.7	73.7	qwen3_5	100.0	99.6
fun_asr_nano_encoder	2.9	2.9	qwen3_5_moe	100.0	99.9
funnel	100.0	100.0	qwen3_5_moe_text	100.0	100.0
fuyu	100.0	100.0	qwen3_5_moe_vision	99.4	91.1
gemma	100.0	100.0	qwen3_5_text	100.0	100.0
gemma2	100.0	100.0	qwen3_asr	99.8	99.8
gemma3	99.7	99.7	qwen3_asr_encoder	100.0	100.0
gemma3_text	100.0	100.0	qwen3_moe	100.0	100.0
gemma3n_audio	83.4	83.4	qwen3_next	100.0	100.0
gemma3n_text	98.1	63.3	qwen3_omni_moe_thinker	99.9	99.1
gemma4_audio	99.0	99.0	qwen3_vl_moe	100.0	98.9
gemma4_vision	100.0	100.0	qwen3_vl_moe_text	100.0	100.0
git	100.0	100.0	qwen3_vl_text	100.0	100.0
glm	100.0	100.0	radio	99.8	99.8
glm4	100.0	100.0	recurrent_gemma	81.2	81.2
glm46v	100.0	100.0	regnet	100.0	100.0
glm4_moe	100.0	100.0	rembert	57.8	51.4
glm4_moe_lite	100.0	100.0	resnet	100.0	100.0
glm4v_moe	100.0	100.0	rf_detr	99.6	99.5
glm4v_moe_text	100.0	100.0	roberta	100.0	100.0
glm4v_moe_vision	99.9	79.2	roberta-prelayernorm	100.0	100.0
glm4v_text	100.0	100.0	roc_bert	100.0	89.8
glm5_next	98.5	98.5	roformer	99.9	99.9
glm5_next_text	98.5	98.5	rt_detr	94.9	94.9
glm5_next_vision	100.0	89.2	rt_detr_v2	94.9	94.9
glm_image	100.0	99.7	rwkv	100.0	100.0
glm_image_text	100.0	100.0	sam	97.4	95.8
glm_image_vision	100.0	100.0	sam2	90.7	89.0
glm_image_vqmodel	100.0	15.8	sam2_hiera_det_model	92.7	92.7
glm_moe_dsa	100.0	100.0	sam2_video	80.4	79.0
glm_ocr	100.0	97.1	sam2_vision_model	92.8	92.8
glm_ocr_text	100.0	100.0	sam3	99.8	99.4
glm_ocr_vision	100.0	94.6	sam3_lite_text	98.9	98.3
glmasr	100.0	100.0	sam3_lite_text_text_model	93.1	93.1
glmasr_encoder	100.0	100.0	sam3_tracker	99.8	99.7
glpn	75.0	75.0	sam3_tracker_video	98.8	98.7
got_ocr2	100.0	99.8	sam3_video	99.1	98.7
gpt-sw3	100.0	100.0	sam3_vision_model	100.0	100.0
gpt_bigcode	100.0	100.0	sam3_vit_model	100.0	100.0
gpt_neo	99.7	99.7	sam_hq	97.1	95.5
gpt_neox	100.0	98.5	sam_hq_vision_model	100.0	100.0
gpt_neox_japanese	100.0	100.0	sapiens2	100.0	100.0
gpt_oss	100.0	100.0	seamless_m4t	96.4	96.4
gptj	100.0	96.6	seamless_m4t_v2	96.4	96.4
gradium_phonon_tts	92.0	92.0	seed_oss	100.0	100.0

Type	Full	Shapes	Type	Full	Shapes
granite	100.0	100.0	segformer	75.0	75.0
granite_speech	100.0	100.0	seggpt	100.0	100.0
granite_speech5_ctc	99.5	76.5	sew	100.0	100.0
granite_speech5_encoder	99.5	76.5	sew-d	100.0	100.0
granite_swa	100.0	100.0	siglip	92.4	92.4
granitemoe	100.0	100.0	siglip2	91.7	91.7
granitemoe_swa	100.0	100.0	siglip2_vision_model	91.7	91.7
granitemoehybrid	98.9	98.9	siglip_vision_model	92.4	92.4
grounding-dino	88.5	86.0	smollm3	100.0	100.0
groupvit	85.9	85.9	solar_open	100.0	100.0
gte	100.0	100.0	speech-encoder-decoder	100.0	100.0
helium	100.0	100.0	speech_to_text	100.0	100.0
hgnet_v2	100.0	100.0	speecht5	100.0	99.9
hiera	96.1	96.1	splinter	100.0	100.0
higgs_audio_v2	100.0	99.5	squeezebert	56.3	56.2
hrm_text	100.0	100.0	stablelm	100.0	100.0
hubert	100.0	100.0	starcoder2	100.0	100.0
hunyuan_vl_vision	94.7	94.7	step3p7	100.0	100.0
hy_v3	100.0	100.0	swiftformer	71.7	71.7
hy_v4	100.0	100.0	swin	100.0	100.0
hyperclovax	100.0	100.0	swin2sr	98.3	98.3
hyperclovax_vision_v2	99.5	97.4	swinv2	100.0	100.0
ibert	11.4	11.4	switch_transformers	100.0	45.7
idefics	100.0	100.0	t5	100.0	89.6
idefics2	99.0	99.0	t5gemma	89.5	89.5
idefics3	99.7	99.7	t5gemma2	99.9	99.9
idefics3_vision	100.0	100.0	tapas	100.0	100.0
ijepa	100.0	100.0	textnet	100.0	100.0
imagegpt	99.0	99.0	timesfm	100.0	100.0
inkling_mm_model	100.0	100.0	timesfm2_5	100.0	100.0
inkling_text	100.0	100.0	timesformer	100.0	75.2
instructblip	100.0	100.0	tipsv2	100.0	100.0
internvl	100.0	99.7	tipsv2_text_model	100.0	100.0
internvl_vision	100.0	100.0	tipsv2_vision_model	100.0	100.0
jais2	100.0	100.0	trocr	100.0	100.0
jamba	100.0	100.0	tvp	99.9	99.9
janus	99.8	99.8	udop	99.7	86.2
jetmoe	97.7	97.7	ultrabert	100.0	100.0
jina_embeddings_v3	100.0	54.3	ultravbert	99.9	99.9
kimi_k25	100.0	100.0	umt5	100.0	96.3
kimi_k25_vision	100.0	100.0	unispeech	100.0	100.0
kimi_linear	99.6	99.6	unispeech-sat	100.0	100.0
kolibri1	100.0	100.0	univnet	100.0	100.0
kosmos-2	98.9	98.9	unlimited_ocr	100.0	100.0
kosmos-2.5	99.1	98.7	uvdoc	100.0	100.0
kyutai_speech_to_text	99.7	99.7	vaultgemma	100.0	100.0
laguna	100.0	100.0	vibevoice	98.4	98.2
lasr_ctc	100.0	100.0	vibevoice_acoustic_tokenizer	100.0	100.0
lasr_encoder	100.0	100.0	vibevoice_acoustic_tokenizer_decoder	100.0	100.0
layoutlm	100.0	100.0	vibevoice_acoustic_tokenizer_encoder	100.0	100.0
layoutlmv3	100.0	99.6	vibevoice_asr	100.0	99.7
led	99.8	99.8	video_llama_3	100.0	100.0
levit	100.0	100.0	video_llama_3_vision	100.0	100.0
lfm2	100.0	100.0	video_llava	100.0	100.0
lfm2_vl	99.7	99.7	videomae	100.0	100.0
lightglue	64.0	64.0	videoprism	100.0	100.0
lighton_ocr	99.6	99.6	videoprism_vision_model	100.0	100.0
lilt	99.9	99.3	vilt	100.0	100.0
llama4	100.0	100.0	vipllava	100.0	99.8
llama4_text	100.0	100.0	vision-encoder-decoder	100.0	100.0
llava	100.0	100.0	vision-text-dual-encoder	100.0	100.0
llava_next	100.0	100.0	visual_bert	99.6	99.6
llava_onevision	100.0	100.0	vit	100.0	100.0
longcat_flash	100.0	100.0	vit_mae	100.0	100.0
longformer	100.0	100.0	vitdet	100.0	100.0
longt5	100.0	89.6	vits	72.8	72.8

Type	Full	Shapes	Type	Full	Shapes
luke	99.9	46.8	vivit	100.0	100.0
lw_detr	98.9	95.5	vjepa2	99.9	99.9
lxmert	100.0	100.0	voxtral	100.0	100.0
m2m_100	100.0	100.0	voxtral_realtime	99.9	99.9
mamba2	98.2	98.2	voxtral_realtime_encoder	100.0	100.0
marian	99.5	99.5	wav2vec2	100.0	100.0
markuplm	97.0	91.2	wavlm	100.0	100.0
mask2former	99.9	99.9	whisper	100.0	100.0
maskformer	100.0	100.0	xclip	98.3	81.7
maskformer-swin	100.0	100.0	xcodec	99.3	99.3
mbart	100.0	100.0	xcodec2	100.0	100.0
megatron-bert	100.0	100.0	xglm	100.0	100.0
mellum	100.0	100.0	xlm-roberta-xl	100.0	100.0
mgp-str	100.0	100.0	xlnet	44.1	35.0
mimi	99.5	99.5	xlstm	37.3	34.3
mimi_decoder	100.0	100.0	xmod	69.6	69.6
mimi_encoder	100.0	100.0	yolos	100.0	100.0
mimo_v2_flash	100.0	100.0	yoso	100.0	100.0
minicpm3	100.0	100.0	youtu	100.0	100.0
minicpm_ocr	99.6	99.1	zamba	100.0	100.0
minicpm_ocr_vision	98.3	70.4	zamba2	78.8	78.8
minicpmv4_6	99.6	99.2	zaya	97.5	97.5

F Hub benchmark: all checkpoints

The benchmark page is https://joessattes.github.io/ai-model-explorer/bench.html; its model list is in bench/models.js in the repository, and the export analysed here is research/data/hub-bench-191.jsonl.gz. Table 12 lists the repositories that could not be read and Table 13 the result for each of the 191 that could. Header sizes are those of the original run, before the loader learned to skip duplicate files, so they are larger than necessary for the few repositories that ship several copies of the same weights.

Repository	Group	Reason
CohereLabs/c4ai-command-r7b-12-2024	Dense LM	gated: the license must be accepted on the Hub first
NousResearch/Meta-Llama-3.1-405B	Dense LM	gated: the license must be accepted on the Hub first
ai21labs/AI21-Jamba-Mini-1.7	SSM / hybrid	gated: the license must be accepted on the Hub first
black-forest-labs/FLUX.1-schnell	Image / video gen.	gated: the license must be accepted on the Hub first
google/gemma-3-270m	Dense LM	gated: the license must be accepted on the Hub first
google/paligemma2-3b-pt-224	Vision–language	gated: the license must be accepted on the Hub first
pyannote/segmentation-3.0	Speech / audio	gated: the license must be accepted on the Hub first
Helsinki-NLP/opus-mt-en-de	Seq2seq	old non-zip PyTorch pickle (not read by design)
google/electra-base-discriminator	Encoder	old non-zip PyTorch pickle (not read by design)
facebook/mms-1b-all	Speech / audio	rate limited: one adapter file per language; the loader now skips adapters

Table 12: The ten repositories that could not be read.

Table 13: All 191 checkpoints read from the Hub, by group. *Params* is our logical count and *Hub* the count on the model page; the next column marks agreement (=), a difference (≠, explained in Section 6.6) or a diffusers pipeline whose Hub count equals our denoiser (den). *d* and layers are for the backbone. *Roles* is role accuracy against labels transferred from the corpus, where the model type is in the corpus.

Repository	Storage	Params	Hub		Family	*d*	Layers	Header (KB)	Roles
Dense LM (45)									
`Qwen/Qwen2.5-72B`		72.7 B	72.7 B	=	dec	8192	80	109	100
`deepseek-ai/DeepSeek-R1-Distill-Llama-70B`		70.6 B	70.6 B	=	dec	8192	80	83	100
`unsloth/Llama-3.3-70B-Instruct`		70.6 B	70.6 B	=	dec	8192	80	83	100
`Qwen/QwQ-32B`		32.8 B	32.8 B	=	dec	5120	64	87	100
`Qwen/Qwen3-32B`		32.8 B	32.8 B	=	dec	5120	64	80	100
`unsloth/gemma-3-27b-it`		27.4 B	27.4 B	=	enc	5376	62	163	
`mistralai/Devstral-Small-2505`		23.6 B	23.6 B	=	dec	5120	40	41	100
`mistralai/Mistral-Small-24B-Instruct-2501`		23.6 B	23.6 B	=	dec	5120	40	41	100
`microsoft/phi-4`		14.7 B	14.7 B	=	dec	5120	40	28	100
`mistralai/Mistral-Nemo-Instruct-2407`		12.2 B	12.2 B	=	dec	5120	40	41	100
`THUDM/glm-4-9b-chat-hf`		9.40 B	9.40 B	=	dec	4096	40	50	100
`unsloth/gemma-2-9b`		9.24 B	9.24 B	=	enc	3584	42	53	100
`internlm/internlm3-8b-instruct`		8.80 B	8.80 B	=	dec	4096	48	49	
`Qwen/Qwen3-8B`		8.19 B	8.19 B	=	dec	4096	36	45	100
`ibm-granite/granite-3.3-8b-instruct`		8.17 B	8.17 B	=	enc	4096	40	41	100
`swiss-ai/Apertus-8B-2509`		8.05 B	8.05 B	=	dec	4096	32	49	100
`nvidia/Llama-3.1-Nemotron-Nano-8B-v1`		8.03 B	8.03 B	=	dec	4096	32	33	100
`unsloth/Meta-Llama-3.1-8B`		8.03 B	8.03 B	=	dec	4096	32	33	100
`Qwen/Qwen2.5-7B`		7.62 B	7.62 B	=	dec	3584	28	38	100
`Qwen/Qwen2.5-Coder-7B-Instruct`		7.62 B	7.62 B	=	dec	3584	28	38	100
`deepseek-ai/DeepSeek-R1-Distill-Qwen-7B`		7.62 B	7.62 B	=	dec	3584	28	38	100
`tiiuae/Falcon3-7B-Base`		7.46 B	7.46 B	=	dec	3072	28	29	100
`allenai/OLMo-2-1124-7B`		7.30 B	7.30 B	=	dec	4096	32	40	100
`unsloth/mistral-7b-v0.3`		7.25 B	7.25 B	=	enc	4096	32	33	98
`mistralai/Mistral-7B-v0.1`		7.24 B	7.24 B	=	dec	4096	32	33	100
`tiiuae/falcon-7b`		7.22 B	7.22 B	=	dec	4544	32	23	100
`Qwen/Qwen3-4B`		4.02 B	4.02 B	=	enc	2560	36	45	100
`microsoft/Phi-4-mini-instruct`		3.84 B	3.84 B	=	enc	3072	32	22	100
`microsoft/Phi-3-mini-4k-instruct`		3.82 B	3.82 B	=	dec	3072	32	22	100
`unsloth/Llama-3.2-3B`		3.21 B	3.21 B	=	enc	3072	28	29	100
`HuggingFaceTB/SmolLM3-3B`		3.08 B	3.08 B	=	enc	2048	36	37	100
`microsoft/phi-2`		2.78 B	2.78 B	=	dec	2560	32	49	100
`unsloth/gemma-2-2b`		2.61 B	2.61 B	=	enc	2304	26	33	100
`HuggingFaceTB/SmolLM2-1.7B`		1.71 B	1.71 B	=	enc	2048	24	25	100
`openai-community/gpt2-xl`		1.61 B	1.61 B	=	enc	1600	48	57	
`EleutherAI/pythia-1.4b`		1.52 B	1.52 B	=	dec	2048	24	41	92
`facebook/opt-1.3b`		1.42 B	–		dec	2048	24	136	100
`unsloth/Llama-3.2-1B`		1.24 B	1.24 B	=	enc	2048	16	16	100
`LiquidAI/LFM2-1.2B`		1.17 B	1.17 B	=	enc	2048	16	16	100
`unsloth/gemma-3-1b-it`		1000 M	1000 M	=	enc	1152	26	38	100
`Qwen/Qwen3-0.6B`		752 M	752 M	=	dec	1024	28	35	100
`bigscience/bloom-560m`		559 M	559 M	=	enc	1024	24	30	
`Qwen/Qwen2.5-0.5B`		494 M	494 M	=	enc	896	24	32	100
`baidu/ERNIE-4.5-0.3B-PT`		361 M	361 M	=	enc	1024	18	18	100
`openai-community/gpt2`		137 M	137 M	=	enc	768	12	14	
MoE LM (24)									
`moonshotai/Kimi-K2-Instruct`	FP8	1.03 T	1.03 T	=	dec	7168	61	16.9 M	
`deepseek-ai/DeepSeek-R1`	FP8	684 B	684 B	=	dec	7168	61	11.3 M	
`deepseek-ai/DeepSeek-V3`	FP8	684 B	684 B	=	dec	7168	61	11.3 M	
`deepseek-ai/DeepSeek-V3.1`	FP8	684 B	685 B	=	dec	7168	61	11.3 M	
`meituan-longcat/LongCat-Flash-Chat`		562 B	562 B	=	dec	6144	28	5.3 M	
`Qwen/Qwen3-Coder-480B-A35B-Instruct`		480 B	480 B	=	dec	6144	62	3.6 M	
`zai-org/GLM-4.5`		358 B	358 B	=	dec	5120	92	5.3 M	
`Qwen/Qwen3-235B-A22B`		235 B	235 B	=	dec	4096	94	4.4 M	
`MiniMaxAI/MiniMax-M2`	FP8	229 B	229 B	=	dec	3072	62	12.0 M	
`openai/gpt-oss-120b`	MXFP4	117 B	117 B	=	dec	2880	36	76	100
`zai-org/GLM-4.5-Air`		110 B	110 B	=	dec	4096	46	2.2 M	
`unsloth/Llama-4-Scout-17B-16E-Instruct`		109 B	109 B	=	prefix	5120	48	148	
`Qwen/Qwen3-Next-80B-A3B-Instruct`		81.3 B	81.3 B	=	dec	2048	48	9.0 M	

Repository	Storage	Params	Hub		Family	d	Layers	Header (KB)	Roles
tencent/Hunyuan-A13B-Instruct		80.4 B	80.4 B	=	enc	4096	32	792	
mistralai/Mixtral-8x7B-v0.1		46.7 B	46.7 B	=	dec	4096	32	125	
microsoft/Phi-3.5-MoE-instruct		41.9 B	41.9 B	=	dec	4096	32	244	
Qwen/Qwen3-30B-A3B		30.5 B	30.5 B	=	dec	2048	48	2.2 M	
baidu/ERNIE-4.5-21B-A3B-PT		21.9 B	21.9 B	=	dec	2560	28	667	
openai/gpt-oss-20b	MXFP4	20.9 B	20.9 B	=	dec	2880	24	51	100
moonshotai/Moonlight-16B-A3B		16.0 B	16.0 B	=	dec	2048	27	640	
deepseek-ai/DeepSeek-V2-Lite		15.7 B	15.7 B	=	dec	2048	27	643	
Qwen/Qwen1.5-MoE-A2.7B		14.3 B	14.3 B	=	dec	2048	24	563	
allenai/OLMoE-1B-7B-0924		6.92 B	6.92 B	=	dec	2048	16	389	
ibm-granite/granite-3.1-3b-a800m-instruct		3.30 B	3.30 B	=	enc	1536	32	34	100
SSM / hybrid (9)									
nvidia/NVIDIA-Nemotron-Nano-9B-v2		8.89 B	8.89 B	=	dec	4480	56	37	
nvidia/Nemotron-H-8B-Base-8K		8.10 B	8.10 B	=	dec	4096	52	34	
tiiuae/Falcon-H1-7B-Base		7.59 B	7.59 B	=	dec	3072	44	83	100
tiiuae/falcon-mamba-7b		7.27 B	7.27 B	=	enc	4096	64	70	100
ibm-granite/granite-4.0-h-tiny		6.94 B	6.94 B	=	enc	1536	40	66	
NX-AI/xLSTM-7b		6.87 B	6.87 B	=	dec	4096	32	56	37
state-spaces/mamba-2.8b-hf		2.77 B	2.77 B	=	enc	2560	64	69	100
Zyphra/Zamba2-2.7B		2.66 B	2.66 B	=	enc	128	5	58	4
RWKV/v6-Finch-1B6-HF		1.60 B	–		dec	2048	24	142	
Encoder (7)									
FacebookAI/xlm-roberta-large		561 M	561 M	=	enc	1024	24	48	100
answerdotai/ModernBERT-large		396 M	396 M	=	enc	1024	28	18	100
FacebookAI/roberta-large		355 M	355 M	=	enc	1024	24	47	100
microsoft/deberta-v3-base		186 M	–		enc	768	12	106	100
google-bert/bert-base-uncased		110 M	110 M	=	enc	768	12	23	100
distilbert/distilbert-base-uncased		67.0 M	67.0 M	=	enc	768	6	12	100
albert/albert-base-v2		11.8 M	11.8 M	=	?	–	–	4	45
Embedding (10)									
Qwen/Qwen3-Embedding-8B		7.57 B	7.57 B	=	enc	4096	36	43	
Qwen/Qwen3-Embedding-0.6B		596 M	596 M	=	enc	1024	28	33	
Qwen/Qwen3-Reranker-0.6B		596 M	596 M	=	enc	1024	28	34	100
BAAI/bge-m3		568 M	–		enc	1024	24	138	
intfloat/multilingual-e5-large		560 M	560 M	=	enc	1024	24	44	
BAAI/bge-large-en-v1.5		335 M	335 M	=	enc	1024	24	44	
Alibaba-NLP/gte-multilingual-base		305 M	305 M	=	enc	768	12	15	
nomic-ai/nomic-embed-text-v1.5		137 M	137 M	=	enc	768	12	11	
sentence-transformers/all-mpnet-base-v2		109 M	109 M	=	enc	768	12	21	
sentence-transformers/all-MiniLM-L6-v2		22.7 M	22.7 M	=	enc	384	6	11	
Seq2seq (6)									
facebook/nllb-200-distilled-600M		1.40 B	–		enc-dec	1024	12	160	100
google/mt5-base		967 M	–		prefix	768	12	155	100
facebook/m2m100_418M		877 M	–		enc-dec	1024	12	171	100
google/flan-t5-large		783 M	783 M	=	prefix	1024	24	67	100
facebook/bart-large-cnn		406 M	406 M	=	enc	1024	12	59	100
google-t5/t5-base		223 M	223 M	=	enc	768	12	30	100
Speech / audio (23)									
Qwen/Qwen2.5-Omni-7B		10.7 B	10.7 B	=	diff	3584	28	301	
Qwen/Qwen2-Audio-7B-Instruct		8.40 B	8.40 B	=	prefix	4096	32	104	
kyutai/stt-2.6b-en		2.71 B	2.62 B	$\neq$	enc	2048	48	75	
microsoft/VibeVoice-1.5B		2.70 B	2.70 B	=	diff	1536	28	155	
ResembleAI/chatterbox		2.67 B	–		prefix	1024	30	749	
facebook/seamless-m4t-v2-large		2.31 B	2.31 B	=	enc	1024	24	272	96
nari-labs/Dia-1.6B-0626		1.61 B	1.61 B	=	enc-dec	2048	18	41	100
openai/whisper-large-v3		1.54 B	1.54 B	=	enc	1280	32	293	
openai/whisper-large-v3-turbo		809 M	809 M	=	enc	1280	4	67	100
distil-whisper/distil-large-v3		756 M	756 M	=	enc	1280	2	123	
facebook/musicgen-small		591 M	591 M	=	enc	1024	24	77	
facebook/wav2vec2-large-xlsr-53		317 M	–		enc	1024	24	181	100
facebook/hubert-large-ls960-ft		315 M	–		enc	1024	24	178	100
microsoft/wavlm-large		315 M	–		enc	1024	24	178	
openai/whisper-small		242 M	242 M	=	enc	768	12	54	100
laion/clap-htsat-unfused		154 M	–		enc	768	12	157	100
microsoft/speecht5_tts		146 M	–		enc	768	12	146	100
facebook/wav2vec2-base-960h		94.4 M	94.4 M	=	enc	768	12	25	100

Repository	Storage	Params	Hub		Family	d	Layers	Header (KB)	Roles
MIT/ast-finetuned-audioset-10-10-0.4593		86.6 M	86.6 M	=	enc	768	12	28	
UsefulSensors/moonshine-base		61.5 M	61.5 M	=	enc	416	8	24	100
openai/whisper-tiny		37.8 M	37.8 M	=	enc	384	4	19	100
facebook/encodec_24khz		23.3 M	23.3 M	=	enc	64	4	26	
speechbrain/spkrec-ecapa-voxceleb		20.8 M	–		enc	128	7	126	
Vision (19)									
google/siglip-so400m-patch14-384		878 M	878 M	=	enc	1152	27	106	96
facebook/sam-vit-huge		641 M	641 M	=	enc	1280	32	67	100
openai/clip-vit-large-patch14		428 M	428 M	=	enc	768	12	70	
google/siglip2-base-patch16-224		375 M	375 M	=	enc	768	12	48	92
Intel/dpt-large		342 M	342 M	=	enc	1024	24	54	
depth-anything/Depth-Anything-V2-Large-hf		335 M	335 M	=	enc	1024	24	59	
facebook/dinov2-large		304 M	304 M	=	enc	1024	24	47	
IDEA-Research/grounding-dino-base		233 M	233 M	=	enc	512	24	163	
facebook/sam2.1-hiera-large		224 M	224 M	=	enc	144	48	106	95
facebook/mask2former-swin-large-coco-panoptic		216 M	–		enc	768	24	240	
google/owlv2-base-patch16-ensemble		155 M	155 M	=	enc	512	12	50	
facebook/convnext-base-224		88.6 M	–		enc	512	36	145	
microsoft/swin-base-patch4-window7-224		87.8 M	87.8 M	=	enc	512	24	55	
google/vit-base-patch16-224		86.6 M	86.6 M	=	enc	768	12	23	
nvidia/segformer-b5-finetuned-ade-640-640		84.7 M	–		enc	320	53	366	
PekingU/rtdetr_r50vd		43.0 M	43.0 M	=	enc	256	48	96	84
facebook/detr-resnet-50		41.6 M	41.6 M	=	enc	256	13	62	
microsoft/resnet-50		25.6 M	25.6 M	=	enc	256	48	41	100
timm/efficientnet_b0.ra_in1k		5.33 M	5.33 M	=	enc	480	16	32	
Vision–language (18)									
Qwen/Qwen3-VL-235B-A22B-Instruct		236 B	236 B	=	prefix	4096	94	174	
zai-org/GLM-4.5V		108 B	108 B	=	prefix	4096	46	2.4 M	
Qwen/Qwen2.5-VL-72B-Instruct		73.4 B	73.4 B	=	prefix	8192	80	151	
Qwen/Qwen3-VL-30B-A3B-Instruct		31.1 B	31.1 B	=	prefix	2048	48	107	
moonshotai/Kimi-VL-A3B-Instruct		16.4 B	16.4 B	=	prefix	2048	27	765	
nvidia/NVIDIA-Nemotron-Nano-12B-v2-VL-BF16		13.2 B	13.2 B	=	prefix	5120	62	95	
mistral-community/pixtral-12b		12.7 B	12.7 B	=	prefix	5120	40	76	
unsloth/Llama-3.2-11B-Vision-Instruct		10.7 B	10.7 B	=	enc-dec	4096	40	116	
Qwen/Qwen2.5-VL-7B-Instruct		8.29 B	8.29 B	=	prefix	3584	28	79	
llava-hf/llava-onevision-qwen2-7b-ov-hf		8.03 B	8.03 B	=	prefix	3584	28	99	
allenai/Molmo-7B-D-0924		8.02 B	8.02 B	=	prefix	3584	28	79	
OpenGVLab/InternVL3-8B-hf		7.94 B	7.94 B	=	prefix	3584	28	97	
llava-hf/llava-1.5-7b-hf		7.06 B	7.06 B	=	prefix	4096	32	90	
microsoft/Phi-4-multimodal-instruct		6.40 B	5.57 B	$\neq$	enc	3072	32	368	
deepseek-ai/Janus-Pro-7B		4.99 B	–		enc	4096	20	249	
unsloth/gemma-3-4b-it		4.30 B	4.30 B	=	enc	2560	34	116	
HuggingFaceTB/SmolVLM2-2.2B-Instruct		2.25 B	2.25 B	=	prefix	2048	24	83	
lerobot/smolvla_base		450 M	450 M	$\neq$	prefix	960	16	72	
Image / video gen. (14)									
Qwen/Qwen-Image		28.8 B	20.4 B	den	diff	3584	28	321	
HiDream-ai/HiDream-I1-Full		22.8 B	17.1 B	den	enc	4096	24	338	
hunyuanvideo-community/HunyuanVideo		20.7 B	12.8 B	den	diff	4096	32	229	
Wan-AI/Wan2.2-TI2V-5B-Diffusers		11.4 B	5.00 B	den	enc	4096	24	134	
Wan-AI/Wan2.1-T2V-1.3B-Diffusers		7.23 B	1.42 B	den	enc	4096	24	132	
Lightricks/LTX-Video		7.10 B	1.92 B	den	diff	4096	24	2.9 M	
THUDM/CogVideoX-2b		6.67 B	1.69 B	den	enc	4096	24	159	
PixArt-alpha/PixArt-Sigma-XL-2-1024-MS		5.46 B	611 M	den	diff	4096	24	120	
Efficient-Large-Model/Sana_1600M_1024px_diffusers		5.32 B	–		diff	2304	26	282	
playgroundai/playground-v2.5-1024px-aesthetic		3.47 B	2.57 B	den	enc-dec	1280	32	1.3 M	
stabilityai/sdxl-turbo		3.47 B	2.57 B	den	enc-dec	1280	32	1.4 M	
stabilityai/stable-diffusion-xl-base-1.0		3.47 B	2.57 B	den	enc-dec	1280	32	1.8 M	
stable-diffusion-v1-5/stable-diffusion-v1-5		1.37 B	860 M	den	enc	768	12	846	
facebook/DiT-XL-2-256		833 M	–		enc	1152	28	346	
Time series / tabular (6)									
google/timesfm-2.0-500m-pytorch		499 M	499 M	=	enc	1280	50	89	
amazon/chronos-bolt-base		205 M	205 M	=	enc	768	12	31	100
Salesforce/moirai-1.1-R-base		91.4 M	91.4 M	=	enc	768	12	18	
amazon/chronos-t5-small		46.2 M	46.2 M	=	enc	512	6	15	95
Prior-Labs/TabPFN-v2-clf		7.24 M	–		enc	768	12	92	
ibm-granite/granite-timeseries-ttm-r2		805 K	805 K	=	enc	48	6	19	

Repository	Storage	Params	Hub		Family	d	Layers	Header (KB)	Roles
Quantized release (10)									
`nvidia/DeepSeek-R1-0528-FP4`	NVFP4	684 B	397 B	=	dec	7168	61	20.4 M	
`Qwen/Qwen3-30B-A3B-GPTQ-Int4`	GPTQ	30.5 B	30.5 B	=	dec	2048	48	8.7 M	
`Qwen/Qwen3-8B-FP8`	FP8	8.19 B	8.19 B	=	dec	4096	36	76	100
`mlx-community/Qwen3-8B-4bit`	MLX	8.19 B	8.19 B	=	dec	4096	36	101	
`nvidia/Qwen3-8B-FP4`	NVFP4	8.19 B	4.72 B	$\neq$	dec	4096	36	131	
`unsloth/Qwen3-8B-bnb-4bit`	NF4	8.19 B	8.41 B	$\neq$	dec	4096	36	193	
`RedHatAI/Meta-Llama-3.1-8B-Instruct-FP8`	FP8	8.03 B	8.03 B	=	dec	4096	32	83	
`nvidia/Llama-3.1-8B-Instruct-FP4`	NVFP4	8.03 B	4.54 B	$\neq$	dec	4096	32	109	
`Qwen/Qwen2.5-7B-Instruct-AWQ`	AWQ	7.62 B	7.62 B	=	dec	3584	28	80	
`unsloth/Llama-3.2-1B-Instruct-bnb-4bit`	NF4	1.24 B	1.27 B	$\neq$	enc	2048	16	81	

G Additional interface views

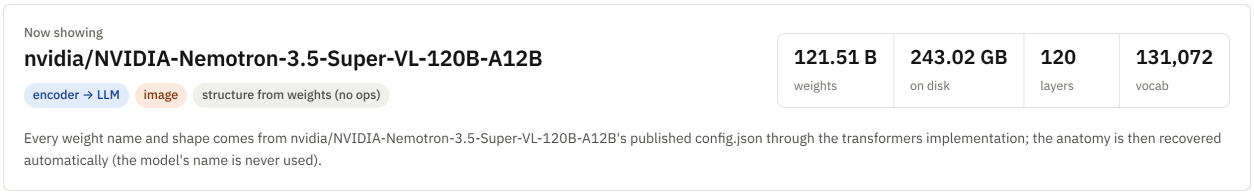


Figure 7: Summary strip for Nemotron 3.5 Super VL: weights, file size, layers and vocabulary, with the family and modality tags.

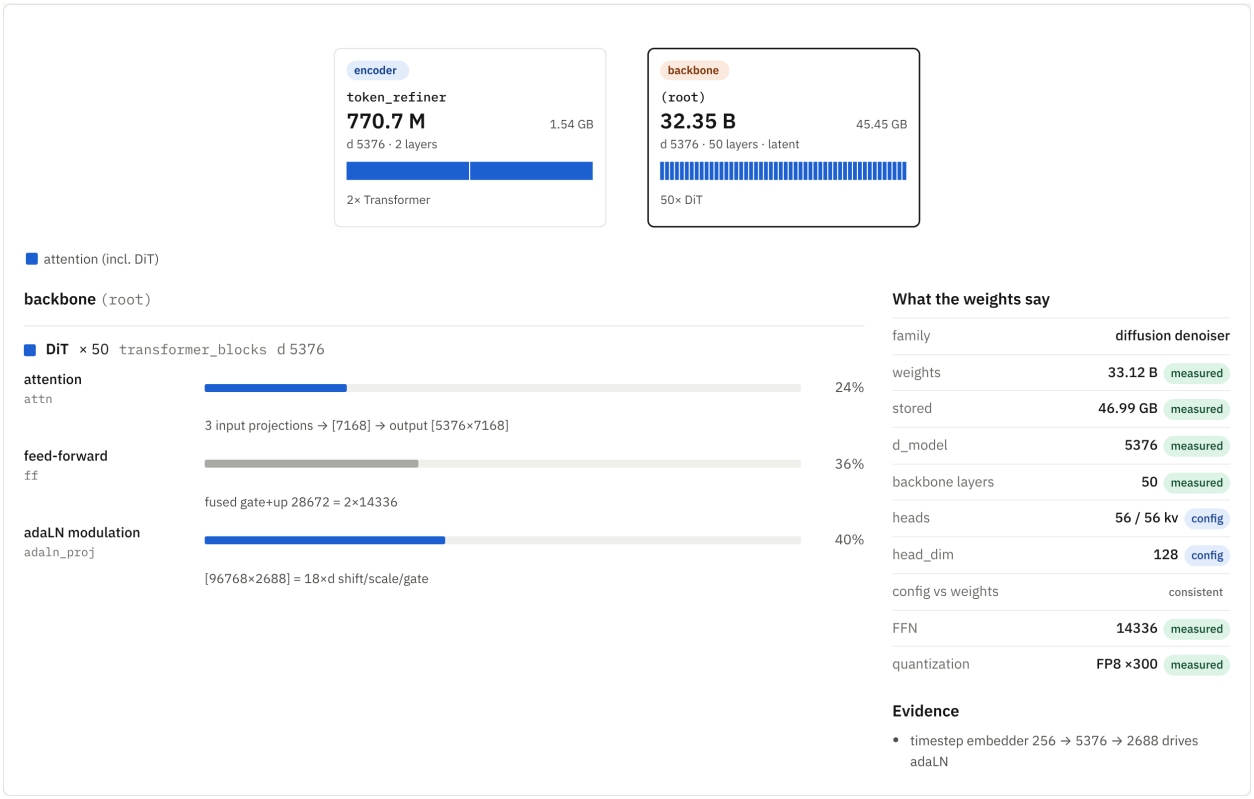


Figure 8: Anatomy view of the FastH3 Omni FP8 denoiser (reconstructed manifest): a two-block token refiner and a 50-block DiT backbone. The head geometry is measured from per-head norms, and the FP8 storage of 300 tensors is shown with logical and stored sizes.

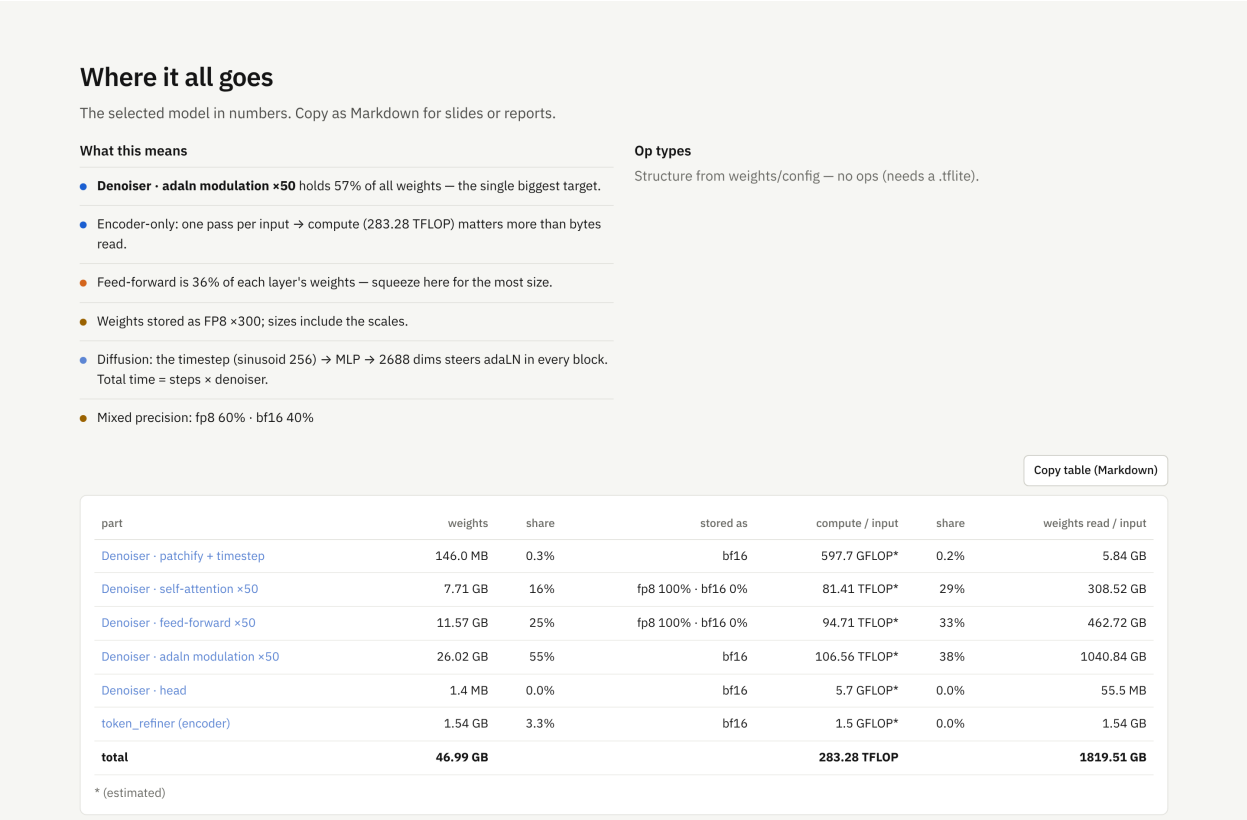


Figure 9: Report view for the same model: where parameters, compute and memory traffic go, with generated observations.