```
      args = f"SERVER_BENCH_N_PROMPTS={args.n_prompts} SERVER_BENCH_MAX_PROMPT_TOKENS={args.max_prompt_tokens}
SERVER_BENCH_MAX_CONTEXT={args.max_tokens} "
    args = args + ' '.join([str(arg) for arg in [k6_path, *k6_args]])
    print(f"bench: starting k6 with: {args}")
    k6_completed = subprocess.run(args, shell=True, stdout=sys.stdout, stderr=sys.stderr)
    if k6_completed.returncode != 0:
        raise Exception("bench: unable to run k6")


def start_server(args):
    server_process = start_server_background(args)

    attempts = 0
    max_attempts = 600
    if 'GITHUB_ACTIONS' in os.environ:
        max_attempts *= 2

    while not is_server_listening(args.host, args.port):
        attempts += 1
        if attempts > max_attempts:
            assert False, "server not started"
        print(f"bench:     waiting for server to start ...")
        time.sleep(0.5)

    attempts = 0
    while not is_server_ready(args.host, args.port):
        attempts += 1
        if attempts > max_attempts:
            assert False, "server not ready"
        print(f"bench:     waiting for server to be ready ...")
        time.sleep(0.5)

    print("bench: server started and ready.")
    return server_process


def start_server_background(args):
    # Start the server
    server_path = '../../../build/bin/llama-server'
    if 'LLAMA_SERVER_BIN_PATH' in os.environ:
        server_path = os.environ['LLAMA_SERVER_BIN_PATH']
    server_args = [
        '--host', args.host,
        '--port', args.port,
    ]
    server_args.extend(['--hf-repo', args.hf_repo])
    server_args.extend(['--hf-file', args.hf_file])
    server_args.extend(['--n-gpu-layers', args.n_gpu_layers])
    server_args.extend(['--ctx-size', args.ctx_size])
    server_args.extend(['--parallel', args.parallel])
    server_args.extend(['--batch-size', args.batch_size])
    server_args.extend(['--ubatch-size', args.ubatch_size])
    server_args.extend(['--n-predict', args.max_tokens * 2])
    server_args.extend(['--defrag-thold', "0.1"])
```

```
    server_args.append('--cont-batching')
    server_args.append('--metrics')
    server_args.append('--flash-attn')
    args = [str(arg) for arg in [server_path, *server_args]]
    print(f"bench: starting server with: {' '.join(args)}")
    pkwargs = {
        'stdout': subprocess.PIPE,
        'stderr': subprocess.PIPE
    }
    server_process = subprocess.Popen(
        args,
        **pkwargs)  # pyright: ignore[reportArgumentType, reportCallIssue]

    def server_log(in_stream, out_stream):
        for line in iter(in_stream.readline, b''):
            print(line.decode('utf-8'), end='', file=out_stream)

    thread_stdout = threading.Thread(target=server_log, args=(server_process.stdout, sys.stdout))
    thread_stdout.start()
    thread_stderr = threading.Thread(target=server_log, args=(server_process.stderr, sys.stderr))
    thread_stderr.start()

    return server_process


def is_server_listening(server_fqdn, server_port):
    with closing(socket.socket(socket.AF_INET, socket.SOCK_STREAM)) as sock:
        result = sock.connect_ex((server_fqdn, server_port))
        _is_server_listening = result == 0
        if _is_server_listening:
            print(f"server is listening on {server_fqdn}:{server_port}...")
        return _is_server_listening


def is_server_ready(server_fqdn, server_port):
    url = f"http://{server_fqdn}:{server_port}/health"
    response = requests.get(url)
    return response.status_code == 200


def escape_metric_name(metric_name):
    return re.sub('[^A-Z0-9]', '_', metric_name.upper())


if __name__ == '__main__':
    main()

==== benchmark_agent.py ====

import time
import random
import psutil
import threading
```

```
results = {}

def simulate_fragment_walks(num_fragments, walk_speed_per_sec):
    walks_done = 0
    start_time = time.time()
    end_time = start_time + 10
    while time.time() < end_time:
        walks_done += walk_speed_per_sec
        time.sleep(1)
    results['walks'] = walks_done

def simulate_mutation_ops(rate_per_sec):
    mutations_done = 0
    start_time = time.time()
    end_time = start_time + 10
    while time.time() < end_time:
        mutations_done += rate_per_sec
        time.sleep(1)
    results['mutations'] = mutations_done

def simulate_emotion_decay_ops(fragments_count, decay_passes_per_sec):
    decay_ops_done = 0
    start_time = time.time()
    end_time = start_time + 10
    while time.time() < end_time:
        decay_ops_done += decay_passes_per_sec
        time.sleep(1)
    results['decay'] = decay_ops_done

def run():
    walk_thread = threading.Thread(target=simulate_fragment_walks, args=(10000, random.randint(200, 350)))
    mutate_thread = threading.Thread(target=simulate_mutation_ops, args=(random.randint(30, 60),))
    decay_thread = threading.Thread(target=simulate_emotion_decay_ops, args=(10000, random.randint(50, 100)))

    walk_thread.start()
    mutate_thread.start()
    decay_thread.start()

    walk_thread.join()
    mutate_thread.join()
    decay_thread.join()

    results['cpu_usage_percent'] = psutil.cpu_percent(interval=1)
    results['ram_usage_percent'] = psutil.virtual_memory().percent

    print("===== Symbolic TPS Benchmark =====")
    print(f"Fragment Walks     : {results['walks'] // 10} per second")
    print(f"Mutations          : {results['mutations'] // 10} per second")
    print(f"Emotion Decay Ops  : {results['decay'] // 10} per second")
    print()
    print(f"CPU Usage          : {results['cpu_usage_percent']}%")
    print(f"RAM Usage          : {results['ram_usage_percent']}%")
    print("==================================")
```

```
if __name__ == "__main__":
    run()

==== boot_wrapper.py ====
import subprocess
import os
import platform
import time
import psutil
from pathlib import Path

SCRIPTS = [
    "deep_system_scan.py",
    "auto_configurator.py",
    "path_optimizer.py",
    "fragment_teleporter.py",
    "run_logicshredder.py",
    "decay_scheduler.py",
    "decay_listener.py"
]

LOG_PATH = Path("logs/boot_times.log")
LOG_PATH.parent.mkdir(exist_ok=True)

def run_script(name, timings):
    if not Path(name).exists():
        print(f"[boot] ? Missing script: {name}")
        timings.append((name, "MISSING", "-", "-"))
        return False

    print(f"[boot] ? Running: {name}")
    start = time.time()
    proc = psutil.Popen(["python", name])

    peak_mem = 0
    cpu_percent = []

    try:
        while proc.is_running():
            mem = proc.memory_info().rss / (1024**2)
            peak_mem = max(peak_mem, mem)
            cpu = proc.cpu_percent(interval=0.1)
            cpu_percent.append(cpu)
    except Exception:
        pass

    end = time.time()
    duration = round(end - start, 2)
    avg_cpu = round(sum(cpu_percent) / len(cpu_percent), 1) if cpu_percent else 0

    print(f"[boot] ? {name} finished in {duration}s | CPU: {avg_cpu}% | MEM: {int(peak_mem)}MB")
    timings.append((name, duration, avg_cpu, int(peak_mem)))
    return proc.returncode == 0
```

```
def log_timings(timings, total):
    with open(LOG_PATH, "a", encoding="utf-8") as log:
        log.write(f"\n=== BOOT TELEMETRY [{time.strftime('%Y-%m-%d %H:%M:%S')}] ===\n")
        for name, dur, cpu, mem in timings:
            log.write(f" - {name}: {dur}s | CPU: {cpu}% | MEM: {mem}MB\n")
        log.write(f"TOTAL BOOT TIME: {round(total, 2)} seconds\n")

def main():
    print("? LOGICSHREDDER SYSTEM BOOT STARTED")
    print(f"? Platform: {platform.system()} | Python: {platform.python_version()}")
    print("==============================================\n")

    start_total = time.time()
    timings = []

    for script in SCRIPTS:
        success = run_script(script, timings)
        if not success:
            print(f"[boot] ? Boot aborted due to failure in {script}")
            break

    total_time = time.time() - start_total
    print(f"[OK] BOOT COMPLETE in {round(total_time, 2)} seconds.")
    log_timings(timings, total_time)

if __name__ == "__main__":
    main()

==== build_structure.py ====
"""
LOGICSHREDDER :: build_structure.py
Purpose: Create the full file/folder structure for the LOGICSHREDDER project
"""

from pathlib import Path

BASE = Path(".")  # run from root folder like: python build_structure.py

DIRS = [
    "agents",
    "core",
    "fragments/core",
    "fragments/incoming",
    "fragments/archive",
    "fragments/overflow",
    "fragments/cold",
    "input",
    "logs",
    "logs/agent_stats",
    "quant/models",
    "snapshots",
    "configs",
    "utils"
]
```

```
FILES = [
    "run_logicshredder.py",
    "neuro_lock.py",
    "start_logicshredder.bat",
    "start_logicshredder_silent.bat",
    "inject_profiler.py",
    "build_structure.py"
]

def make_dirs():
    for d in DIRS:
        path = BASE / d
        path.mkdir(parents=True, exist_ok=True)
        keep = path / ".gitkeep"
        keep.touch()
        print(f"[structure] ? Created: {d}/")

def make_files():
    for f in FILES:
        path = BASE / f
        if not path.exists():
            path.write_text("# Auto-created placeholder\n")
            print(f"[structure] ? Created placeholder: {f}")

if __name__ == "__main__":
    print("CONFIG Building LOGICSHREDDER file structure...")
    make_dirs()
    make_files()
    print("[OK] Project structure initialized.")

==== cold_logic_mover.py ====
from core.config_loader import get
"""
LOGICSHREDDER :: cold_logic_mover.py
Purpose: Move stale, low-confidence beliefs from core to cold storage
"""

import os
import time
import yaml
import threading
from utils import agent_profiler
# [PROFILER_INJECTED]
threading.Thread(target=agent_profiler.run_profile_loop, daemon=True).start()
from pathlib import Path
import shutil

FRAG_DIR = Path("fragments/core")
COLD_DIR = Path("fragments/cold")
LOG_PATH = Path("logs/cold_mover.log")
CONFIDENCE_THRESHOLD = get('tuning.cold_logic_threshold', 0.3)0.3
EMOTION_WEIGHT_PENALTY = 0.15
```

```
FRAG_DIR.mkdir(parents=True, exist_ok=True)
COLD_DIR.mkdir(parents=True, exist_ok=True)
LOG_PATH.parent.mkdir(parents=True, exist_ok=True)

def is_stale(fragment):
    confidence = fragment.get('confidence', 0.5)
    emotion = fragment.get('emotion', {})
    emotion_sum = sum(emotion.values()) if isinstance(emotion, dict) else 0.0
    effective_score = confidence - (emotion_sum * EMOTION_WEIGHT_PENALTY)
    return effective_score < CONFIDENCE_THRESHOLD

def log_cold_move(frag_id, from_path, to_path):
    with open(LOG_PATH, 'a', encoding='utf-8') as log:
        log.write(f"[{int(time.time())}] COLD MOVE: {frag_id} -> {to_path.name}\n")

def move_stale_beliefs():
    files = list(FRAG_DIR.glob("*.yaml"))
    for path in files:
        try:
            with open(path, 'r', encoding='utf-8') as file:
                frag = yaml.safe_load(file)
                if frag and is_stale(frag):
                    target = COLD_DIR / path.name
                    shutil.move(str(path), target)
                    log_cold_move(frag.get('id', path.stem), path, target)
                    print(f"[cold_logic_mover] Archived: {path.name}")
        except Exception as e:
            print(f"[cold_logic_mover] Error on {path.name}: {e}")

if __name__ == "__main__":
    while True:
        move_stale_beliefs()
        time.sleep(10)  # Sweep every 10 seconds
# [CONFIG_PATCHED]

==== combine_to_pdf.py ====
# combine_to_pdf.py
# Converts any supported file (.txt, .md, .py, .pdf, .jpg, .png, etc.) into a PDF

import sys
from pathlib import Path
from fpdf import FPDF
from PyPDF2 import PdfMerger
from PIL import Image

TEXT_EXTS = {'.txt', '.md', '.py'}
IMAGE_EXTS = {'.png', '.jpg', '.jpeg', '.bmp', '.gif'}
PDF_EXT = '.pdf'

def convert_text_to_pdf(txt_path, output_path):
    pdf = FPDF()
    pdf.set_auto_page_break(auto=True, margin=15)
    pdf.add_page()
    pdf.set_font("Courier", size=10)
```

```
    with open(txt_path, "r", encoding="utf-8") as f:
        for line in f:
            pdf.multi_cell(0, 10, line.strip())
    pdf.output(output_path)

def convert_image_to_pdf(img_path, output_path):
    image = Image.open(img_path).convert("RGB")
    image.save(output_path)

def convert_to_pdf(file_path):
    path = Path(file_path)
    ext = path.suffix.lower()
    output_path = path.with_suffix('.pdf')

    if ext in TEXT_EXTS:
        convert_text_to_pdf(path, output_path)
    elif ext in IMAGE_EXTS:
        convert_image_to_pdf(path, output_path)
    elif ext == PDF_EXT:
        return path  # Already PDF
    else:
        raise Exception(f"Unsupported format: {file_path}")

    return output_path

if __name__ == "__main__":
    if len(sys.argv) < 2:
        print("Usage: python combine_to_pdf.py file1 [file2 ...]")
        sys.exit(1)

    for file in sys.argv[1:]:
        try:
            pdf_path = convert_to_pdf(file)
            print(f"[?] Converted: {file} -> {pdf_path}")
        except Exception as e:
            print(f"[?] {file} failed: {e}")



==== compare-llama-bench.py ====
#!/usr/bin/env python3

import logging
import argparse
import heapq
import sys
import os
from glob import glob
import sqlite3

try:
    import git
    from tabulate import tabulate
except ImportError as e:
```

```
    print("the following Python libraries are required: GitPython, tabulate.") # noqa: NP100
    raise e

logger = logging.getLogger("compare-llama-bench")

# Properties by which to differentiate results per commit:
KEY_PROPERTIES = [
    "cpu_info", "gpu_info", "backends", "n_gpu_layers", "model_filename", "model_type", "n_batch", "n_ubatch",
         "embeddings",  "cpu_mask",  "cpu_strict",  "poll",  "n_threads",  "type_k",  "type_v",  "use_mmap",
"no_kv_offload",
    "split_mode", "main_gpu", "tensor_split", "flash_attn", "n_prompt", "n_gen"
]

# Properties that are boolean and are converted to Yes/No for the table:
BOOL_PROPERTIES = ["embeddings", "cpu_strict", "use_mmap", "no_kv_offload", "flash_attn"]

# Header names for the table:
PRETTY_NAMES = {
    "cpu_info": "CPU", "gpu_info": "GPU", "backends": "Backends", "n_gpu_layers": "GPU layers",
    "model_filename": "File", "model_type": "Model", "model_size": "Model size [GiB]",
    "model_n_params": "Num. of par.", "n_batch": "Batch size", "n_ubatch": "Microbatch size",
    "embeddings": "Embeddings", "cpu_mask": "CPU mask", "cpu_strict": "CPU strict", "poll": "Poll",
     "n_threads": "Threads", "type_k": "K type", "type_v": "V type", "split_mode": "Split mode", "main_gpu":
"Main GPU",
     "no_kv_offload": "NKVO", "flash_attn": "FlashAttention", "tensor_split": "Tensor split", "use_mmap": "Use
mmap",
}

DEFAULT_SHOW = ["model_type"]  # Always show these properties by default.
DEFAULT_HIDE = ["model_filename"]  # Always hide these properties by default.
GPU_NAME_STRIP = ["NVIDIA GeForce ", "Tesla ", "AMD Radeon "]  # Strip prefixes for smaller tables.
MODEL_SUFFIX_REPLACE = {" - Small": "_S", " - Medium": "_M", " - Large": "_L"}

DESCRIPTION = """Creates tables from llama-bench data written to an SQLite database. Example usage (Linux):

$ git checkout master
$ make clean && make llama-bench
$ ./llama-bench -o sql | sqlite3 llama-bench.sqlite
$ git checkout some_branch
$ make clean && make llama-bench
$ ./llama-bench -o sql | sqlite3 llama-bench.sqlite
$ ./scripts/compare-llama-bench.py

Performance numbers from multiple runs per commit are averaged WITHOUT being weighted by the --repetitions
parameter of llama-bench.
"""

parser = argparse.ArgumentParser(
    description=DESCRIPTION, formatter_class=argparse.RawDescriptionHelpFormatter)
help_b = (
    "The baseline commit to compare performance to. "
    "Accepts either a branch name, tag name, or commit hash. "
    "Defaults to latest master commit with data."
)
```

```
parser.add_argument("-b", "--baseline", help=help_b)
help_c = (
    "The commit whose performance is to be compared to the baseline. "
    "Accepts either a branch name, tag name, or commit hash. "
    "Defaults to the non-master commit for which llama-bench was run most recently."
)
parser.add_argument("-c", "--compare", help=help_c)
help_i = (
    "Input SQLite file for comparing commits. "
    "Defaults to 'llama-bench.sqlite' in the current working directory. "
    "If no such file is found and there is exactly one .sqlite file in the current directory, "
    "that file is instead used as input."
)
parser.add_argument("-i", "--input", help=help_i)
help_o = (
    "Output format for the table. "
    "Defaults to 'pipe' (GitHub compatible). "
    "Also supports e.g. 'latex' or 'mediawiki'. "
    "See tabulate documentation for full list."
)
parser.add_argument("-o", "--output", help=help_o, default="pipe")
help_s = (
    "Columns to add to the table. "
    "Accepts a comma-separated list of values. "
    f"Legal values: {', '.join(KEY_PROPERTIES[:-2])}. "
    "Defaults to model name (model_type) and CPU and/or GPU name (cpu_info, gpu_info) "
    "plus any column where not all data points are the same. "
    "If the columns are manually specified, then the results for each unique combination of the "
    "specified values are averaged WITHOUT weighing by the --repetitions parameter of llama-bench."
)
parser.add_argument("--check",  action="store_true",  help="check  if  all  required  Python  libraries  are
installed")
parser.add_argument("-s", "--show", help=help_s)
parser.add_argument("--verbose", action="store_true", help="increase output verbosity")

known_args, unknown_args = parser.parse_known_args()

logging.basicConfig(level=logging.DEBUG if known_args.verbose else logging.INFO)

if known_args.check:
    # Check if all required Python libraries are installed. Would have failed earlier if not.
    sys.exit(0)

if unknown_args:
    logger.error(f"Received unknown args: {unknown_args}.\n")
    parser.print_help()
    sys.exit(1)

input_file = known_args.input
if input_file is None and os.path.exists("./llama-bench.sqlite"):
    input_file = "llama-bench.sqlite"
if input_file is None:
    sqlite_files = glob("*.sqlite")
    if len(sqlite_files) == 1:
```

```
        input_file = sqlite_files[0]

if input_file is None:
    logger.error("Cannot find a suitable input file, please provide one.\n")
    parser.print_help()
    sys.exit(1)

connection = sqlite3.connect(input_file)
cursor = connection.cursor()

build_len_min: int = cursor.execute("SELECT MIN(LENGTH(build_commit)) from test;").fetchone()[0]
build_len_max: int = cursor.execute("SELECT MAX(LENGTH(build_commit)) from test;").fetchone()[0]

if build_len_min != build_len_max:
     logger.warning(f"{input_file} contains commit hashes of differing lengths. It's possible that the wrong
commits will be compared. "
                    "Try purging the the database of old commits.")
    cursor.execute(f"UPDATE test SET build_commit = SUBSTRING(build_commit, 1, {build_len_min});")

build_len: int = build_len_min

builds = cursor.execute("SELECT DISTINCT build_commit FROM test;").fetchall()
builds = list(map(lambda b: b[0], builds))  # list[tuple[str]] -> list[str]

if not builds:
    raise RuntimeError(f"{input_file} does not contain any builds.")

try:
    repo = git.Repo(".", search_parent_directories=True)
except git.InvalidGitRepositoryError:
    repo = None


def find_parent_in_data(commit: git.Commit):
    """Helper function to find the most recent parent measured in number of commits for which there is data."""
    heap: list[tuple[int, git.Commit]] = [(0, commit)]
    seen_hexsha8 = set()
    while heap:
        depth, current_commit = heapq.heappop(heap)
        current_hexsha8 = commit.hexsha[:build_len]
        if current_hexsha8 in builds:
            return current_hexsha8
        for parent in commit.parents:
            parent_hexsha8 = parent.hexsha[:build_len]
            if parent_hexsha8 not in seen_hexsha8:
                seen_hexsha8.add(parent_hexsha8)
                heapq.heappush(heap, (depth + 1, parent))
    return None


def get_all_parent_hexsha8s(commit: git.Commit):
    """Helper function to recursively get hexsha8 values for all parents of a commit."""
    unvisited = [commit]
    visited   = []
```

```
    while unvisited:
        current_commit = unvisited.pop(0)
        visited.append(current_commit.hexsha[:build_len])
        for parent in current_commit.parents:
            if parent.hexsha[:build_len] not in visited:
                unvisited.append(parent)

    return visited


def get_commit_name(hexsha8: str):
    """Helper function to find a human-readable name for a commit if possible."""
    if repo is None:
        return hexsha8
    for h in repo.heads:
        if h.commit.hexsha[:build_len] == hexsha8:
            return h.name
    for t in repo.tags:
        if t.commit.hexsha[:build_len] == hexsha8:
            return t.name
    return hexsha8


def get_commit_hexsha8(name: str):
    """Helper function to search for a commit given a human-readable name."""
    if repo is None:
        return None
    for h in repo.heads:
        if h.name == name:
            return h.commit.hexsha[:build_len]
    for t in repo.tags:
        if t.name == name:
            return t.commit.hexsha[:build_len]
    for c in repo.iter_commits("--all"):
        if c.hexsha[:build_len] == name[:build_len]:
            return c.hexsha[:build_len]
    return None


hexsha8_baseline = name_baseline = None

# If the user specified a baseline, try to find a commit for it:
if known_args.baseline is not None:
    if known_args.baseline in builds:
        hexsha8_baseline = known_args.baseline
    if hexsha8_baseline is None:
        hexsha8_baseline = get_commit_hexsha8(known_args.baseline)
        name_baseline = known_args.baseline
    if hexsha8_baseline is None:
        logger.error(f"cannot find data for baseline={known_args.baseline}.")
        sys.exit(1)
# Otherwise, search for the most recent parent of master for which there is data:
elif repo is not None:
```

```
    hexsha8_baseline = find_parent_in_data(repo.heads.master.commit)

    if hexsha8_baseline is None:
        logger.error("No baseline was provided and did not find data for any master branch commits.\n")
        parser.print_help()
        sys.exit(1)
else:
    logger.error("No baseline was provided and the current working directory "
                 "is not part of a git repository from which a baseline could be inferred.\n")
    parser.print_help()
    sys.exit(1)


name_baseline = get_commit_name(hexsha8_baseline)

hexsha8_compare = name_compare = None

# If the user has specified a compare value, try to find a corresponding commit:
if known_args.compare is not None:
    if known_args.compare in builds:
        hexsha8_compare = known_args.compare
    if hexsha8_compare is None:
        hexsha8_compare = get_commit_hexsha8(known_args.compare)
        name_compare = known_args.compare
    if hexsha8_compare is None:
        logger.error(f"cannot find data for compare={known_args.compare}.")
        sys.exit(1)
# Otherwise, search for the commit for llama-bench was most recently run
# and that is not a parent of master:
elif repo is not None:
    hexsha8s_master = get_all_parent_hexsha8s(repo.heads.master.commit)
    builds_timestamp = cursor.execute(
        "SELECT build_commit, test_time FROM test ORDER BY test_time;").fetchall()
    for (hexsha8, _) in reversed(builds_timestamp):
        if hexsha8 not in hexsha8s_master:
            hexsha8_compare = hexsha8
            break

    if hexsha8_compare is None:
        logger.error("No compare target was provided and did not find data for any non-master commits.\n")
        parser.print_help()
        sys.exit(1)
else:
    logger.error("No compare target was provided and the current working directory "
                 "is not part of a git repository from which a compare target could be inferred.\n")
    parser.print_help()
    sys.exit(1)

name_compare = get_commit_name(hexsha8_compare)


def get_rows(properties):
    """
    Helper function that gets table rows for some list of properties.
```

```
    Rows are created by combining those where all provided properties are equal.
    The resulting rows are then grouped by the provided properties and the t/s values are averaged.
    The returned rows are unique in terms of property combinations.
    """
    select_string = ", ".join(
        [f"tb.{p}" for p in properties] + ["tb.n_prompt", "tb.n_gen", "AVG(tb.avg_ts)", "AVG(tc.avg_ts)"])
    equal_string = " AND ".join(
        [f"tb.{p} = tc.{p}" for p in KEY_PROPERTIES] + [
            f"tb.build_commit = '{hexsha8_baseline}'", f"tc.build_commit = '{hexsha8_compare}'"]
    )
    group_order_string = ", ".join([f"tb.{p}" for p in properties] + ["tb.n_gen", "tb.n_prompt"])
    query = (f"SELECT {select_string} FROM test tb JOIN test tc ON {equal_string} "
             f"GROUP BY {group_order_string} ORDER BY {group_order_string};")
    return cursor.execute(query).fetchall()


# If the user provided columns to group the results by, use them:
if known_args.show is not None:
    show = known_args.show.split(",")
    unknown_cols = []
    for prop in show:
        if prop not in KEY_PROPERTIES[:-2]:  # Last two values are n_prompt, n_gen.
            unknown_cols.append(prop)
    if unknown_cols:
        logger.error(f"Unknown values for --show: {', '.join(unknown_cols)}")
        parser.print_usage()
        sys.exit(1)
    rows_show = get_rows(show)
# Otherwise, select those columns where the values are not all the same:
else:
    rows_full = get_rows(KEY_PROPERTIES)
    properties_different = []
    for i, kp_i in enumerate(KEY_PROPERTIES):
        if kp_i in DEFAULT_SHOW or kp_i == "n_prompt" or kp_i == "n_gen":
            continue
        for row_full in rows_full:
            if row_full[i] != rows_full[0][i]:
                properties_different.append(kp_i)
                break

    show = []
    # Show CPU and/or GPU by default even if the hardware for all results is the same:
    if "n_gpu_layers" not in properties_different:
        ngl = int(rows_full[0][KEY_PROPERTIES.index("n_gpu_layers")])

        if ngl != 99 and "cpu_info" not in properties_different:
            show.append("cpu_info")

    show += properties_different

    index_default = 0
    for prop in ["cpu_info", "gpu_info", "n_gpu_layers", "main_gpu"]:
        if prop in show:
            index_default += 1
```

```
    show = show[:index_default] + DEFAULT_SHOW + show[index_default:]
    for prop in DEFAULT_HIDE:
        try:
            show.remove(prop)
        except ValueError:
            pass
    rows_show = get_rows(show)

table = []
for row in rows_show:
    n_prompt = int(row[-4])
    n_gen    = int(row[-3])
    if n_prompt != 0 and n_gen == 0:
        test_name = f"pp{n_prompt}"
    elif n_prompt == 0 and n_gen != 0:
        test_name = f"tg{n_gen}"
    else:
        test_name = f"pp{n_prompt}+tg{n_gen}"
    #           Regular columns    test name    avg t/s values              Speedup
    #            VVVVVVVVVVVVV     VVVVVVVVV    VVVVVVVVVVVVVV              VVVVVVV
    table.append(list(row[:-4]) + [test_name] + list(row[-2:]) + [float(row[-1]) / float(row[-2])])

# Some a-posteriori fixes to make the table contents prettier:
for bool_property in BOOL_PROPERTIES:
    if bool_property in show:
        ip = show.index(bool_property)
        for row_table in table:
            row_table[ip] = "Yes" if int(row_table[ip]) == 1 else "No"

if "model_type" in show:
    ip = show.index("model_type")
    for (old, new) in MODEL_SUFFIX_REPLACE.items():
        for row_table in table:
            row_table[ip] = row_table[ip].replace(old, new)

if "model_size" in show:
    ip = show.index("model_size")
    for row_table in table:
        row_table[ip] = float(row_table[ip]) / 1024 ** 3

if "gpu_info" in show:
    ip = show.index("gpu_info")
    for row_table in table:
        for gns in GPU_NAME_STRIP:
            row_table[ip] = row_table[ip].replace(gns, "")

        gpu_names = row_table[ip].split("/")
        num_gpus = len(gpu_names)
        all_names_the_same = len(set(gpu_names)) == 1
        if len(gpu_names) >= 2 and all_names_the_same:
            row_table[ip] = f"{num_gpus}x {gpu_names[0]}"

headers  = [PRETTY_NAMES[p] for p in show]
headers += ["Test", f"t/s {name_baseline}", f"t/s {name_compare}", "Speedup"]
```

```
print(tabulate( # noqa: NP100
    table,
    headers=headers,
    floatfmt=".2f",
    tablefmt=known_args.output
))

==== compile_to_pdf.py ====
import os
from pathlib import Path
from fpdf import FPDF

# Extensions to include
FILE_EXTENSIONS = [".py", ".yaml", ".yml", ".json", ".txt"]

class CodePDF(FPDF):
    def __init__(self):
        super().__init__()
        self.set_auto_page_break(auto=True, margin=15)
        self.add_page()
        self.set_font("Courier", size=8)

    def add_code_file(self, filepath):
        self.set_font("Courier", size=8)
        self.multi_cell(0, 5, f"\n==== {filepath} ====\n")
        try:
            with open(filepath, 'r', encoding='utf-8', errors='ignore') as f:
                for line in f:
                    clean_line = ''.join(c if 0x20 <= ord(c) <= 0x7E or c in '\t\n\r' else '?' for c in line)
                    self.multi_cell(0, 5, clean_line.rstrip())
        except Exception as e:
            self.multi_cell(0, 5, f"[Error reading {filepath}: {e}]\n")

def gather_files(root_dir, extensions):
    return [
        f for f in Path(root_dir).rglob("*")
         if f.is_file() and f.suffix.lower() in extensions and "venv" not in f.parts and "__pycache__" not in
f.parts
    ]

def main(root=".", output="symbolic_manifesto.pdf"):
    pdf = CodePDF()
    files = gather_files(root, FILE_EXTENSIONS)

    if not files:
        print("[!] No matching files found.")
        return

    for file in sorted(files):
        pdf.add_code_file(file)

    pdf.output(output)
    print(f"[?] Compiled {len(files)} files into: {output}")
```

```
if __name__ == "__main__":
    main()

==== config_access.py ====
# core/config_access.py
import yaml
from pathlib import Path

CONFIG_PATH = Path("configs/system_config.yaml")
_config_cache = {}

def load_config():
    global _config_cache
    if _config_cache:
        return _config_cache
    try:
        with open(CONFIG_PATH, 'r', encoding='utf-8') as f:
            _config_cache = yaml.safe_load(f)
            return _config_cache
    except Exception as e:
        print(f"[config_loader] ERROR Failed to load config: {e}")
        return {}

def get(path, default=None):
    keys = path.split(".")
    val = load_config()
    for key in keys:
        if isinstance(val, dict):
            val = val.get(key)
        else:
            return default
    return val if val is not None else default

==== config_loader.py ====
from core.config_loader import get
"""
LOGICSHREDDER :: config_loader.py
Purpose: Load and access system_config.yaml safely and globally
"""

import yaml
from pathlib import Path

CONFIG_PATH = Path("configs/system_config.yaml")
_config_cache = {}

def load_config():
    global _config_cache
    if _config_cache:
        return _config_cache
    try:
        with open(CONFIG_PATH, 'r', encoding='utf-8') as f:
            _config_cache = yaml.safe_load(f)
```

```
            return _config_cache
    except Exception as e:
        print(f"[config_loader] ERROR Failed to load config: {e}")
        return {}

def get(path, default=None):
    """Get config value by dot.path string (e.g. 'tuning.decay_rate')"""
    keys = path.split(".")
    val = load_config()
    for key in keys:
        if isinstance(val, dict):
            val = val.get(key)
        else:
            return default
    return val if val is not None else default

# [CONFIG_PATCHED]

==== conftest.py ====
import pytest
from utils import *


# ref: https://stackoverflow.com/questions/22627659/run-code-before-and-after-each-test-in-py-test
@pytest.fixture(autouse=True)
def stop_server_after_each_test():
    # do nothing before each test
    yield
    # stop all servers after each test
    instances = set(
        server_instances
    )  # copy the set to prevent 'Set changed size during iteration'
    for server in instances:
        server.stop()

==== constants.py ====
from __future__ import annotations

from enum import Enum, IntEnum, auto
from typing import Any

#
# constants
#

GGUF_MAGIC             = 0x46554747  # "GGUF"
GGUF_VERSION           = 3
GGUF_DEFAULT_ALIGNMENT = 32
GGML_QUANT_VERSION     = 2  # GGML_QNT_VERSION from ggml.h

#
# metadata keys
#
```

```
class Keys:
    class General:
        TYPE                      = "general.type"
        ARCHITECTURE              = "general.architecture"
        QUANTIZATION_VERSION      = "general.quantization_version"
        ALIGNMENT                 = "general.alignment"
        FILE_TYPE                 = "general.file_type"

        # Authorship Metadata
        NAME                      = "general.name"
        AUTHOR                    = "general.author"
        VERSION                   = "general.version"
        ORGANIZATION              = "general.organization"

        FINETUNE                  = "general.finetune"
        BASENAME                  = "general.basename"

        DESCRIPTION               = "general.description"
        QUANTIZED_BY              = "general.quantized_by"

        SIZE_LABEL                = "general.size_label"

        # Licensing details
        LICENSE                   = "general.license"
        LICENSE_NAME              = "general.license.name"
        LICENSE_LINK              = "general.license.link"

        # Typically represents the converted GGUF repo (Unless native)
        URL                       = "general.url" # Model Website/Paper
        DOI                       = "general.doi"
        UUID                      = "general.uuid"
        REPO_URL                  = "general.repo_url" # Model Source Repository (git/svn/etc...)

        # Model Source during conversion
        SOURCE_URL                = "general.source.url" # Model Website/Paper
        SOURCE_DOI                = "general.source.doi"
        SOURCE_UUID               = "general.source.uuid"
        SOURCE_REPO_URL           = "general.source.repo_url" # Model Source Repository (git/svn/etc...)

        # Base Model Source. There can be more than one source if it's a merged
        # model like with 'Mistral-7B-Merge-14-v0.1'. This will assist in
        # tracing linage of models as it is finetuned or merged over time.
        BASE_MODEL_COUNT          = "general.base_model.count"
        BASE_MODEL_NAME           = "general.base_model.{id}.name"
        BASE_MODEL_AUTHOR         = "general.base_model.{id}.author"
        BASE_MODEL_VERSION        = "general.base_model.{id}.version"
        BASE_MODEL_ORGANIZATION   = "general.base_model.{id}.organization"
        BASE_MODEL_DESCRIPTION    = "general.base_model.{id}.description"
        BASE_MODEL_URL            = "general.base_model.{id}.url" # Model Website/Paper
        BASE_MODEL_DOI            = "general.base_model.{id}.doi"
        BASE_MODEL_UUID           = "general.base_model.{id}.uuid"
             BASE_MODEL_REPO_URL            =  "general.base_model.{id}.repo_url"  #  Model  Source  Repository
(git/svn/etc...)
```

```
    # Dataset Source
    DATASET_COUNT           = "general.dataset.count"
    DATASET_NAME            = "general.dataset.{id}.name"
    DATASET_AUTHOR          = "general.dataset.{id}.author"
    DATASET_VERSION         = "general.dataset.{id}.version"
    DATASET_ORGANIZATION    = "general.dataset.{id}.organization"
    DATASET_DESCRIPTION     = "general.dataset.{id}.description"
    DATASET_URL             = "general.dataset.{id}.url" # Model Website/Paper
    DATASET_DOI             = "general.dataset.{id}.doi"
    DATASET_UUID            = "general.dataset.{id}.uuid"
    DATASET_REPO_URL        = "general.dataset.{id}.repo_url" # Model Source Repository (git/svn/etc...)

    # Array based KV stores
    TAGS                        = "general.tags"
    LANGUAGES                   = "general.languages"

class LLM:
    VOCAB_SIZE                        = "{arch}.vocab_size"
    CONTEXT_LENGTH                    = "{arch}.context_length"
    EMBEDDING_LENGTH                  = "{arch}.embedding_length"
    FEATURES_LENGTH                   = "{arch}.features_length"
    BLOCK_COUNT                       = "{arch}.block_count"
    LEADING_DENSE_BLOCK_COUNT         = "{arch}.leading_dense_block_count"
    FEED_FORWARD_LENGTH               = "{arch}.feed_forward_length"
    EXPERT_FEED_FORWARD_LENGTH        = "{arch}.expert_feed_forward_length"
    EXPERT_SHARED_FEED_FORWARD_LENGTH = "{arch}.expert_shared_feed_forward_length"
    USE_PARALLEL_RESIDUAL             = "{arch}.use_parallel_residual"
    TENSOR_DATA_LAYOUT                = "{arch}.tensor_data_layout"
    EXPERT_COUNT                      = "{arch}.expert_count"
    EXPERT_USED_COUNT                 = "{arch}.expert_used_count"
    EXPERT_SHARED_COUNT               = "{arch}.expert_shared_count"
    EXPERT_WEIGHTS_SCALE              = "{arch}.expert_weights_scale"
    EXPERT_WEIGHTS_NORM               = "{arch}.expert_weights_norm"
    EXPERT_GATING_FUNC                = "{arch}.expert_gating_func"
    POOLING_TYPE                      = "{arch}.pooling_type"
    LOGIT_SCALE                       = "{arch}.logit_scale"
    DECODER_START_TOKEN_ID            = "{arch}.decoder_start_token_id"
    ATTN_LOGIT_SOFTCAPPING            = "{arch}.attn_logit_softcapping"
    FINAL_LOGIT_SOFTCAPPING           = "{arch}.final_logit_softcapping"
    SWIN_NORM                         = "{arch}.swin_norm"
    RESCALE_EVERY_N_LAYERS            = "{arch}.rescale_every_n_layers"
    TIME_MIX_EXTRA_DIM                = "{arch}.time_mix_extra_dim"
    TIME_DECAY_EXTRA_DIM              = "{arch}.time_decay_extra_dim"
    RESIDUAL_SCALE                    = "{arch}.residual_scale"
    EMBEDDING_SCALE                   = "{arch}.embedding_scale"
    TOKEN_SHIFT_COUNT                 = "{arch}.token_shift_count"
    INTERLEAVE_MOE_LAYER_STEP         = "{arch}.interleave_moe_layer_step"

class Attention:
    HEAD_COUNT                  = "{arch}.attention.head_count"
    HEAD_COUNT_KV               = "{arch}.attention.head_count_kv"
    MAX_ALIBI_BIAS              = "{arch}.attention.max_alibi_bias"
    CLAMP_KQV                   = "{arch}.attention.clamp_kqv"
```

```
    KEY_LENGTH                  = "{arch}.attention.key_length"
    VALUE_LENGTH                = "{arch}.attention.value_length"
    LAYERNORM_EPS               = "{arch}.attention.layer_norm_epsilon"
    LAYERNORM_RMS_EPS           = "{arch}.attention.layer_norm_rms_epsilon"
    GROUPNORM_EPS               = "{arch}.attention.group_norm_epsilon"
    GROUPNORM_GROUPS            = "{arch}.attention.group_norm_groups"
    CAUSAL                      = "{arch}.attention.causal"
    Q_LORA_RANK                 = "{arch}.attention.q_lora_rank"
    KV_LORA_RANK                = "{arch}.attention.kv_lora_rank"
    DECAY_LORA_RANK             = "{arch}.attention.decay_lora_rank"
    ICLR_LORA_RANK              = "{arch}.attention.iclr_lora_rank"
    VALUE_RESIDUAL_MIX_LORA_RANK = "{arch}.attention.value_residual_mix_lora_rank"
    GATE_LORA_RANK              = "{arch}.attention.gate_lora_rank"
    REL_BUCKETS_COUNT           = "{arch}.attention.relative_buckets_count"
    SLIDING_WINDOW              = "{arch}.attention.sliding_window"
    SCALE                       = "{arch}.attention.scale"

class Rope:
    DIMENSION_COUNT         = "{arch}.rope.dimension_count"
    DIMENSION_SECTIONS      = "{arch}.rope.dimension_sections"
    FREQ_BASE               = "{arch}.rope.freq_base"
    SCALING_TYPE            = "{arch}.rope.scaling.type"
    SCALING_FACTOR          = "{arch}.rope.scaling.factor"
    SCALING_ATTN_FACTOR     = "{arch}.rope.scaling.attn_factor"
    SCALING_ORIG_CTX_LEN    = "{arch}.rope.scaling.original_context_length"
    SCALING_FINETUNED       = "{arch}.rope.scaling.finetuned"
    SCALING_YARN_LOG_MUL    = "{arch}.rope.scaling.yarn_log_multiplier"

class Split:
    LLM_KV_SPLIT_NO            = "split.no"
    LLM_KV_SPLIT_COUNT         = "split.count"
    LLM_KV_SPLIT_TENSORS_COUNT = "split.tensors.count"

class SSM:
    CONV_KERNEL    = "{arch}.ssm.conv_kernel"
    INNER_SIZE     = "{arch}.ssm.inner_size"
    STATE_SIZE     = "{arch}.ssm.state_size"
    TIME_STEP_RANK = "{arch}.ssm.time_step_rank"
    DT_B_C_RMS     = "{arch}.ssm.dt_b_c_rms"

class WKV:
    HEAD_SIZE = "{arch}.wkv.head_size"

class PosNet:
    EMBEDDING_LENGTH = "{arch}.posnet.embedding_length"
    BLOCK_COUNT      = "{arch}.posnet.block_count"

class ConvNext:
    EMBEDDING_LENGTH = "{arch}.convnext.embedding_length"
    BLOCK_COUNT      = "{arch}.convnext.block_count"

class Tokenizer:
    MODEL                = "tokenizer.ggml.model"
    PRE                  = "tokenizer.ggml.pre"
```

```
        LIST                = "tokenizer.ggml.tokens"
        TOKEN_TYPE          = "tokenizer.ggml.token_type"
        TOKEN_TYPE_COUNT    = "tokenizer.ggml.token_type_count"  # for BERT-style token types
        SCORES              = "tokenizer.ggml.scores"
        MERGES              = "tokenizer.ggml.merges"
        BOS_ID              = "tokenizer.ggml.bos_token_id"
        EOS_ID              = "tokenizer.ggml.eos_token_id"
        EOT_ID              = "tokenizer.ggml.eot_token_id"
        EOM_ID              = "tokenizer.ggml.eom_token_id"
        UNK_ID              = "tokenizer.ggml.unknown_token_id"
        SEP_ID              = "tokenizer.ggml.seperator_token_id"
        PAD_ID              = "tokenizer.ggml.padding_token_id"
        MASK_ID             = "tokenizer.ggml.mask_token_id"
        ADD_BOS             = "tokenizer.ggml.add_bos_token"
        ADD_EOS             = "tokenizer.ggml.add_eos_token"
        ADD_PREFIX          = "tokenizer.ggml.add_space_prefix"
        REMOVE_EXTRA_WS     = "tokenizer.ggml.remove_extra_whitespaces"
        PRECOMPILED_CHARSMAP = "tokenizer.ggml.precompiled_charsmap"
        HF_JSON             = "tokenizer.huggingface.json"
        RWKV                = "tokenizer.rwkv.world"
        CHAT_TEMPLATE       = "tokenizer.chat_template"
        CHAT_TEMPLATE_N     = "tokenizer.chat_template.{name}"
        CHAT_TEMPLATES      = "tokenizer.chat_templates"
        # FIM/Infill special tokens constants
        FIM_PRE_ID          = "tokenizer.ggml.fim_pre_token_id"
        FIM_SUF_ID          = "tokenizer.ggml.fim_suf_token_id"
        FIM_MID_ID          = "tokenizer.ggml.fim_mid_token_id"
        FIM_PAD_ID          = "tokenizer.ggml.fim_pad_token_id"
        FIM_REP_ID          = "tokenizer.ggml.fim_rep_token_id"
        FIM_SEP_ID          = "tokenizer.ggml.fim_sep_token_id"
        # deprecated:
        PREFIX_ID           = "tokenizer.ggml.prefix_token_id"
        SUFFIX_ID           = "tokenizer.ggml.suffix_token_id"
        MIDDLE_ID           = "tokenizer.ggml.middle_token_id"

    class Adapter:
        TYPE       = "adapter.type"
        LORA_ALPHA = "adapter.lora.alpha"

#
# recommended mapping of model tensor names for storage in gguf
#


class GGUFType:
    MODEL   = "model"
    ADAPTER = "adapter"


class MODEL_ARCH(IntEnum):
    LLAMA            = auto()
    LLAMA4           = auto()
    DECI             = auto()
    FALCON           = auto()
```

```
BAICHUAN         = auto()
GROK             = auto()
GPT2             = auto()
GPTJ             = auto()
GPTNEOX          = auto()
MPT              = auto()
STARCODER        = auto()
REFACT           = auto()
BERT             = auto()
NOMIC_BERT       = auto()
JINA_BERT_V2     = auto()
BLOOM            = auto()
STABLELM         = auto()
QWEN             = auto()
QWEN2            = auto()
QWEN2MOE         = auto()
QWEN2VL          = auto()
QWEN3            = auto()
QWEN3MOE         = auto()
PHI2             = auto()
PHI3             = auto()
PHIMOE           = auto()
PLAMO            = auto()
CODESHELL        = auto()
ORION            = auto()
INTERNLM2        = auto()
MINICPM          = auto()
MINICPM3         = auto()
GEMMA            = auto()
GEMMA2           = auto()
GEMMA3           = auto()
STARCODER2       = auto()
RWKV6            = auto()
RWKV6QWEN2       = auto()
RWKV7            = auto()
ARWKV7           = auto()
MAMBA            = auto()
XVERSE           = auto()
COMMAND_R        = auto()
COHERE2          = auto()
DBRX             = auto()
OLMO             = auto()
OLMO2            = auto()
OLMOE            = auto()
OPENELM          = auto()
ARCTIC           = auto()
DEEPSEEK         = auto()
DEEPSEEK2        = auto()
CHATGLM          = auto()
GLM4             = auto()
BITNET           = auto()
T5               = auto()
T5ENCODER        = auto()
JAIS             = auto()
```

```
    NEMOTRON         = auto()
    EXAONE           = auto()
    GRANITE          = auto()
    GRANITE_MOE      = auto()
    CHAMELEON        = auto()
    WAVTOKENIZER_DEC = auto()
    PLM              = auto()
    BAILINGMOE       = auto()


class MODEL_TENSOR(IntEnum):
    TOKEN_EMBD           = auto()
    TOKEN_EMBD_NORM      = auto()
    TOKEN_TYPES          = auto()
    POS_EMBD             = auto()
    OUTPUT               = auto()
    OUTPUT_NORM          = auto()
    ROPE_FREQS           = auto()
    ROPE_FACTORS_LONG    = auto()
    ROPE_FACTORS_SHORT   = auto()
    ATTN_Q               = auto()
    ATTN_K               = auto()
    ATTN_V               = auto()
    ATTN_QKV             = auto()
    ATTN_OUT             = auto()
    ATTN_NORM            = auto()
    ATTN_NORM_2          = auto()
    ATTN_OUT_NORM        = auto()
    ATTN_POST_NORM       = auto()
    ATTN_ROT_EMBD        = auto()
    FFN_GATE_INP         = auto()
    FFN_GATE_INP_SHEXP   = auto()
    FFN_NORM             = auto()
    FFN_PRE_NORM         = auto()
    FFN_POST_NORM        = auto()
    FFN_GATE             = auto()
    FFN_DOWN             = auto()
    FFN_UP               = auto()
    FFN_ACT              = auto()
    FFN_NORM_EXP         = auto()
    FFN_GATE_EXP         = auto()
    FFN_DOWN_EXP         = auto()
    FFN_UP_EXP           = auto()
    FFN_GATE_SHEXP       = auto()
    FFN_DOWN_SHEXP       = auto()
    FFN_UP_SHEXP         = auto()
    FFN_EXP_PROBS_B      = auto()
    ATTN_Q_NORM          = auto()
    ATTN_K_NORM          = auto()
    LAYER_OUT_NORM       = auto()
    SSM_IN               = auto()
    SSM_CONV1D           = auto()
    SSM_X                = auto()
    SSM_DT               = auto()
```

```
SSM_A               = auto()
SSM_D               = auto()
SSM_OUT             = auto()
TIME_MIX_W0         = auto()
TIME_MIX_W1         = auto()
TIME_MIX_W2         = auto()
TIME_MIX_A0         = auto()
TIME_MIX_A1         = auto()
TIME_MIX_A2         = auto()
TIME_MIX_V0         = auto()
TIME_MIX_V1         = auto()
TIME_MIX_V2         = auto()
TIME_MIX_G1         = auto()
TIME_MIX_G2         = auto()
TIME_MIX_K_K        = auto()
TIME_MIX_K_A        = auto()
TIME_MIX_R_K        = auto()
TIME_MIX_LERP_X     = auto()
TIME_MIX_LERP_K     = auto()
TIME_MIX_LERP_V     = auto()
TIME_MIX_LERP_R     = auto()
TIME_MIX_LERP_G     = auto()
TIME_MIX_LERP_FUSED = auto()
TIME_MIX_LERP_W     = auto()
TIME_MIX_FIRST      = auto()
TIME_MIX_DECAY      = auto()
TIME_MIX_DECAY_W1   = auto()
TIME_MIX_DECAY_W2   = auto()
TIME_MIX_KEY        = auto()
TIME_MIX_VALUE      = auto()
TIME_MIX_RECEPTANCE = auto()
TIME_MIX_GATE       = auto()
TIME_MIX_LN         = auto()
TIME_MIX_OUTPUT     = auto()
CHANNEL_MIX_LERP_K  = auto()
CHANNEL_MIX_LERP_R  = auto()
CHANNEL_MIX_KEY     = auto()
CHANNEL_MIX_RECEPTANCE = auto()
CHANNEL_MIX_VALUE   = auto()
ATTN_Q_A            = auto()
ATTN_Q_B            = auto()
ATTN_KV_A_MQA       = auto()
ATTN_KV_B           = auto()
ATTN_Q_A_NORM       = auto()
ATTN_KV_A_NORM      = auto()
FFN_SUB_NORM        = auto()
ATTN_SUB_NORM       = auto()
DEC_ATTN_NORM       = auto()
DEC_ATTN_Q          = auto()
DEC_ATTN_K          = auto()
DEC_ATTN_V          = auto()
DEC_ATTN_OUT        = auto()
DEC_ATTN_REL_B      = auto()
DEC_CROSS_ATTN_NORM = auto()
```

```
    DEC_CROSS_ATTN_Q     = auto()
    DEC_CROSS_ATTN_K     = auto()
    DEC_CROSS_ATTN_V     = auto()
    DEC_CROSS_ATTN_OUT   = auto()
    DEC_CROSS_ATTN_REL_B = auto()
    DEC_FFN_NORM         = auto()
    DEC_FFN_GATE         = auto()
    DEC_FFN_DOWN         = auto()
    DEC_FFN_UP           = auto()
    DEC_OUTPUT_NORM      = auto()
    ENC_ATTN_NORM        = auto()
    ENC_ATTN_Q           = auto()
    ENC_ATTN_K           = auto()
    ENC_ATTN_V           = auto()
    ENC_ATTN_OUT         = auto()
    ENC_ATTN_REL_B       = auto()
    ENC_FFN_NORM         = auto()
    ENC_FFN_GATE         = auto()
    ENC_FFN_DOWN         = auto()
    ENC_FFN_UP           = auto()
    ENC_OUTPUT_NORM      = auto()
    CLS                  = auto() # classifier
    CLS_OUT              = auto() # classifier output projection
    CONV1D               = auto()
    CONVNEXT_DW          = auto()
    CONVNEXT_NORM        = auto()
    CONVNEXT_PW1         = auto()
    CONVNEXT_PW2         = auto()
    CONVNEXT_GAMMA       = auto()
    POSNET_CONV1         = auto()
    POSNET_CONV2         = auto()
    POSNET_NORM          = auto()
    POSNET_NORM1         = auto()
    POSNET_NORM2         = auto()
    POSNET_ATTN_NORM     = auto()
    POSNET_ATTN_Q        = auto()
    POSNET_ATTN_K        = auto()
    POSNET_ATTN_V        = auto()
    POSNET_ATTN_OUT      = auto()


MODEL_ARCH_NAMES: dict[MODEL_ARCH, str] = {
    MODEL_ARCH.LLAMA:            "llama",
    MODEL_ARCH.LLAMA4:           "llama4",
    MODEL_ARCH.DECI:             "deci",
    MODEL_ARCH.FALCON:           "falcon",
    MODEL_ARCH.BAICHUAN:         "baichuan",
    MODEL_ARCH.GROK:             "grok",
    MODEL_ARCH.GPT2:             "gpt2",
    MODEL_ARCH.GPTJ:             "gptj",
    MODEL_ARCH.GPTNEOX:          "gptneox",
    MODEL_ARCH.MPT:              "mpt",
    MODEL_ARCH.STARCODER:        "starcoder",
    MODEL_ARCH.REFACT:           "refact",
```

```
MODEL_ARCH.BERT:            "bert",
MODEL_ARCH.NOMIC_BERT:      "nomic-bert",
MODEL_ARCH.JINA_BERT_V2:    "jina-bert-v2",
MODEL_ARCH.BLOOM:           "bloom",
MODEL_ARCH.STABLELM:        "stablelm",
MODEL_ARCH.QWEN:            "qwen",
MODEL_ARCH.QWEN2:           "qwen2",
MODEL_ARCH.QWEN2MOE:        "qwen2moe",
MODEL_ARCH.QWEN2VL:         "qwen2vl",
MODEL_ARCH.QWEN3:           "qwen3",
MODEL_ARCH.QWEN3MOE:        "qwen3moe",
MODEL_ARCH.PHI2:            "phi2",
MODEL_ARCH.PHI3:            "phi3",
MODEL_ARCH.PHIMOE:          "phimoe",
MODEL_ARCH.PLAMO:           "plamo",
MODEL_ARCH.CODESHELL:       "codeshell",
MODEL_ARCH.ORION:           "orion",
MODEL_ARCH.INTERNLM2:       "internlm2",
MODEL_ARCH.MINICPM:         "minicpm",
MODEL_ARCH.MINICPM3:        "minicpm3",
MODEL_ARCH.GEMMA:           "gemma",
MODEL_ARCH.GEMMA2:          "gemma2",
MODEL_ARCH.GEMMA3:          "gemma3",
MODEL_ARCH.STARCODER2:      "starcoder2",
MODEL_ARCH.RWKV6:           "rwkv6",
MODEL_ARCH.RWKV6QWEN2:      "rwkv6qwen2",
MODEL_ARCH.RWKV7:           "rwkv7",
MODEL_ARCH.ARWKV7:          "arwkv7",
MODEL_ARCH.MAMBA:           "mamba",
MODEL_ARCH.XVERSE:          "xverse",
MODEL_ARCH.COMMAND_R:       "command-r",
MODEL_ARCH.COHERE2:         "cohere2",
MODEL_ARCH.DBRX:            "dbrx",
MODEL_ARCH.OLMO:            "olmo",
MODEL_ARCH.OLMO2:           "olmo2",
MODEL_ARCH.OLMOE:           "olmoe",
MODEL_ARCH.OPENELM:         "openelm",
MODEL_ARCH.ARCTIC:          "arctic",
MODEL_ARCH.DEEPSEEK:        "deepseek",
MODEL_ARCH.DEEPSEEK2:       "deepseek2",
MODEL_ARCH.CHATGLM:         "chatglm",
MODEL_ARCH.GLM4:            "glm4",
MODEL_ARCH.BITNET:          "bitnet",
MODEL_ARCH.T5:              "t5",
MODEL_ARCH.T5ENCODER:       "t5encoder",
MODEL_ARCH.JAIS:            "jais",
MODEL_ARCH.NEMOTRON:        "nemotron",
MODEL_ARCH.EXAONE:          "exaone",
MODEL_ARCH.GRANITE:         "granite",
MODEL_ARCH.GRANITE_MOE:     "granitemoe",
MODEL_ARCH.CHAMELEON:       "chameleon",
MODEL_ARCH.WAVTOKENIZER_DEC: "wavtokenizer-dec",
MODEL_ARCH.PLM:             "plm",
MODEL_ARCH.BAILINGMOE:      "bailingmoe",
```

```
}

TENSOR_NAMES: dict[MODEL_TENSOR, str] = {
    MODEL_TENSOR.TOKEN_EMBD:                "token_embd",
    MODEL_TENSOR.TOKEN_EMBD_NORM:           "token_embd_norm",
    MODEL_TENSOR.TOKEN_TYPES:               "token_types",
    MODEL_TENSOR.POS_EMBD:                  "position_embd",
    MODEL_TENSOR.OUTPUT_NORM:               "output_norm",
    MODEL_TENSOR.OUTPUT:                    "output",
    MODEL_TENSOR.ROPE_FREQS:                "rope_freqs",
    MODEL_TENSOR.ROPE_FACTORS_LONG:         "rope_factors_long",
    MODEL_TENSOR.ROPE_FACTORS_SHORT:        "rope_factors_short",
    MODEL_TENSOR.ATTN_NORM:                 "blk.{bid}.attn_norm",
    MODEL_TENSOR.ATTN_NORM_2:               "blk.{bid}.attn_norm_2",
    MODEL_TENSOR.ATTN_QKV:                  "blk.{bid}.attn_qkv",
    MODEL_TENSOR.ATTN_Q:                    "blk.{bid}.attn_q",
    MODEL_TENSOR.ATTN_K:                    "blk.{bid}.attn_k",
    MODEL_TENSOR.ATTN_V:                    "blk.{bid}.attn_v",
    MODEL_TENSOR.ATTN_OUT:                  "blk.{bid}.attn_output",
    MODEL_TENSOR.ATTN_ROT_EMBD:             "blk.{bid}.attn_rot_embd",
    MODEL_TENSOR.ATTN_Q_NORM:               "blk.{bid}.attn_q_norm",
    MODEL_TENSOR.ATTN_K_NORM:               "blk.{bid}.attn_k_norm",
    MODEL_TENSOR.ATTN_OUT_NORM:             "blk.{bid}.attn_output_norm",
    MODEL_TENSOR.ATTN_POST_NORM:            "blk.{bid}.post_attention_norm",
    MODEL_TENSOR.FFN_GATE_INP:              "blk.{bid}.ffn_gate_inp",
    MODEL_TENSOR.FFN_GATE_INP_SHEXP:        "blk.{bid}.ffn_gate_inp_shexp",
    MODEL_TENSOR.FFN_NORM:                  "blk.{bid}.ffn_norm",
    MODEL_TENSOR.FFN_PRE_NORM:              "blk.{bid}.ffn_norm",
    MODEL_TENSOR.FFN_POST_NORM:             "blk.{bid}.post_ffw_norm",
    MODEL_TENSOR.FFN_GATE:                  "blk.{bid}.ffn_gate",
    MODEL_TENSOR.FFN_DOWN:                  "blk.{bid}.ffn_down",
    MODEL_TENSOR.FFN_UP:                    "blk.{bid}.ffn_up",
    MODEL_TENSOR.FFN_GATE_SHEXP:            "blk.{bid}.ffn_gate_shexp",
    MODEL_TENSOR.FFN_DOWN_SHEXP:            "blk.{bid}.ffn_down_shexp",
    MODEL_TENSOR.FFN_UP_SHEXP:              "blk.{bid}.ffn_up_shexp",
    MODEL_TENSOR.FFN_ACT:                   "blk.{bid}.ffn",
    MODEL_TENSOR.FFN_NORM_EXP:              "blk.{bid}.ffn_norm_exps",
    MODEL_TENSOR.FFN_GATE_EXP:              "blk.{bid}.ffn_gate_exps",
    MODEL_TENSOR.FFN_DOWN_EXP:              "blk.{bid}.ffn_down_exps",
    MODEL_TENSOR.FFN_UP_EXP:                "blk.{bid}.ffn_up_exps",
    MODEL_TENSOR.FFN_EXP_PROBS_B:           "blk.{bid}.exp_probs_b",
    MODEL_TENSOR.LAYER_OUT_NORM:            "blk.{bid}.layer_output_norm",
    MODEL_TENSOR.SSM_IN:                    "blk.{bid}.ssm_in",
    MODEL_TENSOR.SSM_CONV1D:                "blk.{bid}.ssm_conv1d",
    MODEL_TENSOR.SSM_X:                     "blk.{bid}.ssm_x",
    MODEL_TENSOR.SSM_DT:                    "blk.{bid}.ssm_dt",
    MODEL_TENSOR.SSM_A:                     "blk.{bid}.ssm_a",
    MODEL_TENSOR.SSM_D:                     "blk.{bid}.ssm_d",
    MODEL_TENSOR.SSM_OUT:                   "blk.{bid}.ssm_out",
    MODEL_TENSOR.TIME_MIX_W0:               "blk.{bid}.time_mix_w0",
    MODEL_TENSOR.TIME_MIX_W1:               "blk.{bid}.time_mix_w1",
    MODEL_TENSOR.TIME_MIX_W2:               "blk.{bid}.time_mix_w2",
    MODEL_TENSOR.TIME_MIX_A0:               "blk.{bid}.time_mix_a0",
    MODEL_TENSOR.TIME_MIX_A1:               "blk.{bid}.time_mix_a1",
```

```
MODEL_TENSOR.TIME_MIX_A2:              "blk.{bid}.time_mix_a2",
MODEL_TENSOR.TIME_MIX_V0:              "blk.{bid}.time_mix_v0",
MODEL_TENSOR.TIME_MIX_V1:              "blk.{bid}.time_mix_v1",
MODEL_TENSOR.TIME_MIX_V2:              "blk.{bid}.time_mix_v2",
MODEL_TENSOR.TIME_MIX_G1:              "blk.{bid}.time_mix_g1",
MODEL_TENSOR.TIME_MIX_G2:              "blk.{bid}.time_mix_g2",
MODEL_TENSOR.TIME_MIX_K_K:             "blk.{bid}.time_mix_k_k",
MODEL_TENSOR.TIME_MIX_K_A:             "blk.{bid}.time_mix_k_a",
MODEL_TENSOR.TIME_MIX_R_K:             "blk.{bid}.time_mix_r_k",
MODEL_TENSOR.TIME_MIX_LERP_X:          "blk.{bid}.time_mix_lerp_x",
MODEL_TENSOR.TIME_MIX_LERP_K:          "blk.{bid}.time_mix_lerp_k",
MODEL_TENSOR.TIME_MIX_LERP_V:          "blk.{bid}.time_mix_lerp_v",
MODEL_TENSOR.TIME_MIX_LERP_R:          "blk.{bid}.time_mix_lerp_r",
MODEL_TENSOR.TIME_MIX_LERP_G:          "blk.{bid}.time_mix_lerp_g",
MODEL_TENSOR.TIME_MIX_LERP_FUSED:      "blk.{bid}.time_mix_lerp_fused",
MODEL_TENSOR.TIME_MIX_LERP_W:          "blk.{bid}.time_mix_lerp_w",
MODEL_TENSOR.TIME_MIX_FIRST:           "blk.{bid}.time_mix_first",
MODEL_TENSOR.TIME_MIX_DECAY:           "blk.{bid}.time_mix_decay",
MODEL_TENSOR.TIME_MIX_DECAY_W1:        "blk.{bid}.time_mix_decay_w1",
MODEL_TENSOR.TIME_MIX_DECAY_W2:        "blk.{bid}.time_mix_decay_w2",
MODEL_TENSOR.TIME_MIX_KEY:             "blk.{bid}.time_mix_key",
MODEL_TENSOR.TIME_MIX_VALUE:           "blk.{bid}.time_mix_value",
MODEL_TENSOR.TIME_MIX_RECEPTANCE:      "blk.{bid}.time_mix_receptance",
MODEL_TENSOR.TIME_MIX_GATE:            "blk.{bid}.time_mix_gate",
MODEL_TENSOR.TIME_MIX_LN:              "blk.{bid}.time_mix_ln",
MODEL_TENSOR.TIME_MIX_OUTPUT:          "blk.{bid}.time_mix_output",
MODEL_TENSOR.CHANNEL_MIX_LERP_K:       "blk.{bid}.channel_mix_lerp_k",
MODEL_TENSOR.CHANNEL_MIX_LERP_R:       "blk.{bid}.channel_mix_lerp_r",
MODEL_TENSOR.CHANNEL_MIX_KEY:          "blk.{bid}.channel_mix_key",
MODEL_TENSOR.CHANNEL_MIX_RECEPTANCE:   "blk.{bid}.channel_mix_receptance",
MODEL_TENSOR.CHANNEL_MIX_VALUE:        "blk.{bid}.channel_mix_value",
MODEL_TENSOR.ATTN_Q_A:                 "blk.{bid}.attn_q_a",
MODEL_TENSOR.ATTN_Q_B:                 "blk.{bid}.attn_q_b",
MODEL_TENSOR.ATTN_KV_A_MQA:            "blk.{bid}.attn_kv_a_mqa",
MODEL_TENSOR.ATTN_KV_B:                "blk.{bid}.attn_kv_b",
MODEL_TENSOR.ATTN_Q_A_NORM:            "blk.{bid}.attn_q_a_norm",
MODEL_TENSOR.ATTN_KV_A_NORM:           "blk.{bid}.attn_kv_a_norm",
MODEL_TENSOR.ATTN_SUB_NORM:            "blk.{bid}.attn_sub_norm",
MODEL_TENSOR.FFN_SUB_NORM:             "blk.{bid}.ffn_sub_norm",
MODEL_TENSOR.DEC_ATTN_NORM:            "dec.blk.{bid}.attn_norm",
MODEL_TENSOR.DEC_ATTN_Q:               "dec.blk.{bid}.attn_q",
MODEL_TENSOR.DEC_ATTN_K:               "dec.blk.{bid}.attn_k",
MODEL_TENSOR.DEC_ATTN_V:               "dec.blk.{bid}.attn_v",
MODEL_TENSOR.DEC_ATTN_OUT:             "dec.blk.{bid}.attn_o",
MODEL_TENSOR.DEC_ATTN_REL_B:           "dec.blk.{bid}.attn_rel_b",
MODEL_TENSOR.DEC_CROSS_ATTN_NORM:      "dec.blk.{bid}.cross_attn_norm",
MODEL_TENSOR.DEC_CROSS_ATTN_Q:         "dec.blk.{bid}.cross_attn_q",
MODEL_TENSOR.DEC_CROSS_ATTN_K:         "dec.blk.{bid}.cross_attn_k",
MODEL_TENSOR.DEC_CROSS_ATTN_V:         "dec.blk.{bid}.cross_attn_v",
MODEL_TENSOR.DEC_CROSS_ATTN_OUT:       "dec.blk.{bid}.cross_attn_o",
MODEL_TENSOR.DEC_CROSS_ATTN_REL_B:     "dec.blk.{bid}.cross_attn_rel_b",
MODEL_TENSOR.DEC_FFN_NORM:             "dec.blk.{bid}.ffn_norm",
MODEL_TENSOR.DEC_FFN_GATE:             "dec.blk.{bid}.ffn_gate",
MODEL_TENSOR.DEC_FFN_DOWN:             "dec.blk.{bid}.ffn_down",
```

```
    MODEL_TENSOR.DEC_FFN_UP:                "dec.blk.{bid}.ffn_up",
    MODEL_TENSOR.DEC_OUTPUT_NORM:           "dec.output_norm",
    MODEL_TENSOR.ENC_ATTN_NORM:             "enc.blk.{bid}.attn_norm",
    MODEL_TENSOR.ENC_ATTN_Q:                "enc.blk.{bid}.attn_q",
    MODEL_TENSOR.ENC_ATTN_K:                "enc.blk.{bid}.attn_k",
    MODEL_TENSOR.ENC_ATTN_V:                "enc.blk.{bid}.attn_v",
    MODEL_TENSOR.ENC_ATTN_OUT:              "enc.blk.{bid}.attn_o",
    MODEL_TENSOR.ENC_ATTN_REL_B:            "enc.blk.{bid}.attn_rel_b",
    MODEL_TENSOR.ENC_FFN_NORM:              "enc.blk.{bid}.ffn_norm",
    MODEL_TENSOR.ENC_FFN_GATE:              "enc.blk.{bid}.ffn_gate",
    MODEL_TENSOR.ENC_FFN_DOWN:              "enc.blk.{bid}.ffn_down",
    MODEL_TENSOR.ENC_FFN_UP:                "enc.blk.{bid}.ffn_up",
    MODEL_TENSOR.ENC_OUTPUT_NORM:           "enc.output_norm",
    MODEL_TENSOR.CLS:                       "cls",
    MODEL_TENSOR.CLS_OUT:                   "cls.output",
    MODEL_TENSOR.CONV1D:                    "conv1d",
    MODEL_TENSOR.CONVNEXT_DW:               "convnext.{bid}.dw",
    MODEL_TENSOR.CONVNEXT_NORM:             "convnext.{bid}.norm",
    MODEL_TENSOR.CONVNEXT_PW1:              "convnext.{bid}.pw1",
    MODEL_TENSOR.CONVNEXT_PW2:              "convnext.{bid}.pw2",
    MODEL_TENSOR.CONVNEXT_GAMMA:            "convnext.{bid}.gamma",
    MODEL_TENSOR.POSNET_CONV1:              "posnet.{bid}.conv1",
    MODEL_TENSOR.POSNET_CONV2:              "posnet.{bid}.conv2",
    MODEL_TENSOR.POSNET_NORM:               "posnet.{bid}.norm",
    MODEL_TENSOR.POSNET_NORM1:              "posnet.{bid}.norm1",
    MODEL_TENSOR.POSNET_NORM2:              "posnet.{bid}.norm2",
    MODEL_TENSOR.POSNET_ATTN_NORM:          "posnet.{bid}.attn_norm",
    MODEL_TENSOR.POSNET_ATTN_Q:             "posnet.{bid}.attn_q",
    MODEL_TENSOR.POSNET_ATTN_K:             "posnet.{bid}.attn_k",
    MODEL_TENSOR.POSNET_ATTN_V:             "posnet.{bid}.attn_v",
    MODEL_TENSOR.POSNET_ATTN_OUT:           "posnet.{bid}.attn_output",
}

MODEL_TENSORS: dict[MODEL_ARCH, list[MODEL_TENSOR]] = {
    MODEL_ARCH.LLAMA: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
    ],
```

```
    MODEL_ARCH.LLAMA4: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
        MODEL_TENSOR.FFN_GATE_SHEXP,
        MODEL_TENSOR.FFN_DOWN_SHEXP,
        MODEL_TENSOR.FFN_UP_SHEXP,
    ],
    MODEL_ARCH.DECI: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
    ],
    MODEL_ARCH.GROK: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
```

```
        MODEL_TENSOR.ATTN_OUT_NORM,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
        MODEL_TENSOR.LAYER_OUT_NORM,
    ],
    MODEL_ARCH.GPTNEOX: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.FALCON: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_NORM_2,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.BAICHUAN: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.STARCODER: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.POS_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
```

```
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.BERT: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.TOKEN_EMBD_NORM,
        MODEL_TENSOR.TOKEN_TYPES,
        MODEL_TENSOR.POS_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_OUT_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.LAYER_OUT_NORM,
        MODEL_TENSOR.CLS,
        MODEL_TENSOR.CLS_OUT,
    ],
    MODEL_ARCH.NOMIC_BERT: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.TOKEN_EMBD_NORM,
        MODEL_TENSOR.TOKEN_TYPES,
        MODEL_TENSOR.POS_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_OUT_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.LAYER_OUT_NORM,
    ],
    MODEL_ARCH.JINA_BERT_V2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.TOKEN_EMBD_NORM,
        MODEL_TENSOR.TOKEN_TYPES,
        MODEL_TENSOR.ATTN_NORM_2,
        MODEL_TENSOR.ATTN_OUT_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.LAYER_OUT_NORM,
```

```
        MODEL_TENSOR.CLS,
    ],
    MODEL_ARCH.MPT: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_ACT,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.POS_EMBD,
    ],
    MODEL_ARCH.GPTJ: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.REFACT: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.BLOOM: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.TOKEN_EMBD_NORM,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
```

```
    ],
    MODEL_ARCH.STABLELM: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K_NORM,
    ],
    MODEL_ARCH.QWEN: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.QWEN2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.QWEN2VL: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
```

```
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.QWEN2MOE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
        MODEL_TENSOR.FFN_GATE_INP_SHEXP,
        MODEL_TENSOR.FFN_GATE_SHEXP,
        MODEL_TENSOR.FFN_DOWN_SHEXP,
        MODEL_TENSOR.FFN_UP_SHEXP,
    ],
    MODEL_ARCH.QWEN3: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.QWEN3MOE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
```

```
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
    ],
    MODEL_ARCH.PLAMO: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.GPT2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.POS_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.PHI2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.PHI3: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FACTORS_LONG,
        MODEL_TENSOR.ROPE_FACTORS_SHORT,
        MODEL_TENSOR.ATTN_NORM,
```

```
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.PHIMOE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FACTORS_LONG,
        MODEL_TENSOR.ROPE_FACTORS_SHORT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
    ],
    MODEL_ARCH.CODESHELL: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.POS_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.ORION: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
```

```
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.INTERNLM2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.MINICPM: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ROPE_FACTORS_LONG,
        MODEL_TENSOR.ROPE_FACTORS_SHORT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
    ],
    MODEL_ARCH.MINICPM3: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FACTORS_LONG,
        MODEL_TENSOR.ROPE_FACTORS_SHORT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q_A,
        MODEL_TENSOR.ATTN_Q_B,
        MODEL_TENSOR.ATTN_KV_A_MQA,
        MODEL_TENSOR.ATTN_KV_B,
        MODEL_TENSOR.ATTN_Q_A_NORM,
        MODEL_TENSOR.ATTN_KV_A_NORM,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
```

```
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.GEMMA: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_NORM,
    ],
    MODEL_ARCH.GEMMA2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_POST_NORM,
        MODEL_TENSOR.FFN_PRE_NORM,
        MODEL_TENSOR.FFN_POST_NORM,
    ],
    MODEL_ARCH.GEMMA3: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_POST_NORM,
        MODEL_TENSOR.FFN_PRE_NORM,
        MODEL_TENSOR.FFN_POST_NORM,
    ],
    MODEL_ARCH.STARCODER2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
```

```
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.RWKV6: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.TOKEN_EMBD_NORM,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_NORM_2,
        MODEL_TENSOR.TIME_MIX_W1,
        MODEL_TENSOR.TIME_MIX_W2,
        MODEL_TENSOR.TIME_MIX_LERP_X,
        MODEL_TENSOR.TIME_MIX_LERP_K,
        MODEL_TENSOR.TIME_MIX_LERP_V,
        MODEL_TENSOR.TIME_MIX_LERP_R,
        MODEL_TENSOR.TIME_MIX_LERP_G,
        MODEL_TENSOR.TIME_MIX_LERP_W,
        MODEL_TENSOR.TIME_MIX_LERP_FUSED,
        MODEL_TENSOR.TIME_MIX_FIRST,
        MODEL_TENSOR.TIME_MIX_DECAY,
        MODEL_TENSOR.TIME_MIX_DECAY_W1,
        MODEL_TENSOR.TIME_MIX_DECAY_W2,
        MODEL_TENSOR.TIME_MIX_KEY,
        MODEL_TENSOR.TIME_MIX_VALUE,
        MODEL_TENSOR.TIME_MIX_RECEPTANCE,
        MODEL_TENSOR.TIME_MIX_GATE,
        MODEL_TENSOR.TIME_MIX_LN,
        MODEL_TENSOR.TIME_MIX_OUTPUT,
        MODEL_TENSOR.CHANNEL_MIX_LERP_K,
        MODEL_TENSOR.CHANNEL_MIX_LERP_R,
        MODEL_TENSOR.CHANNEL_MIX_KEY,
        MODEL_TENSOR.CHANNEL_MIX_RECEPTANCE,
        MODEL_TENSOR.CHANNEL_MIX_VALUE,
    ],
    MODEL_ARCH.RWKV6QWEN2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.TIME_MIX_W1,
        MODEL_TENSOR.TIME_MIX_W2,
        MODEL_TENSOR.TIME_MIX_LERP_X,
        MODEL_TENSOR.TIME_MIX_LERP_K,
        MODEL_TENSOR.TIME_MIX_LERP_V,
        MODEL_TENSOR.TIME_MIX_LERP_R,
```

```
        MODEL_TENSOR.TIME_MIX_LERP_G,
        MODEL_TENSOR.TIME_MIX_LERP_W,
        MODEL_TENSOR.TIME_MIX_LERP_FUSED,
        MODEL_TENSOR.TIME_MIX_FIRST,
        MODEL_TENSOR.TIME_MIX_DECAY,
        MODEL_TENSOR.TIME_MIX_DECAY_W1,
        MODEL_TENSOR.TIME_MIX_DECAY_W2,
        MODEL_TENSOR.TIME_MIX_KEY,
        MODEL_TENSOR.TIME_MIX_VALUE,
        MODEL_TENSOR.TIME_MIX_RECEPTANCE,
        MODEL_TENSOR.TIME_MIX_GATE,
        MODEL_TENSOR.TIME_MIX_LN,
        MODEL_TENSOR.TIME_MIX_OUTPUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.RWKV7: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.TOKEN_EMBD_NORM,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_NORM_2,
        MODEL_TENSOR.TIME_MIX_LERP_FUSED,
        MODEL_TENSOR.TIME_MIX_W0,
        MODEL_TENSOR.TIME_MIX_W1,
        MODEL_TENSOR.TIME_MIX_W2,
        MODEL_TENSOR.TIME_MIX_A0,
        MODEL_TENSOR.TIME_MIX_A1,
        MODEL_TENSOR.TIME_MIX_A2,
        MODEL_TENSOR.TIME_MIX_V0,
        MODEL_TENSOR.TIME_MIX_V1,
        MODEL_TENSOR.TIME_MIX_V2,
        MODEL_TENSOR.TIME_MIX_G1,
        MODEL_TENSOR.TIME_MIX_G2,
        MODEL_TENSOR.TIME_MIX_K_K,
        MODEL_TENSOR.TIME_MIX_K_A,
        MODEL_TENSOR.TIME_MIX_R_K,
        MODEL_TENSOR.TIME_MIX_KEY,
        MODEL_TENSOR.TIME_MIX_VALUE,
        MODEL_TENSOR.TIME_MIX_RECEPTANCE,
        MODEL_TENSOR.TIME_MIX_LN,
        MODEL_TENSOR.TIME_MIX_OUTPUT,
        MODEL_TENSOR.CHANNEL_MIX_LERP_K,
        MODEL_TENSOR.CHANNEL_MIX_KEY,
        MODEL_TENSOR.CHANNEL_MIX_VALUE,
    ],
    MODEL_ARCH.ARWKV7: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.TOKEN_EMBD_NORM,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
```

```
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.TIME_MIX_LERP_FUSED,
        MODEL_TENSOR.TIME_MIX_W0,
        MODEL_TENSOR.TIME_MIX_W1,
        MODEL_TENSOR.TIME_MIX_W2,
        MODEL_TENSOR.TIME_MIX_A0,
        MODEL_TENSOR.TIME_MIX_A1,
        MODEL_TENSOR.TIME_MIX_A2,
        MODEL_TENSOR.TIME_MIX_V0,
        MODEL_TENSOR.TIME_MIX_V1,
        MODEL_TENSOR.TIME_MIX_V2,
        MODEL_TENSOR.TIME_MIX_G1,
        MODEL_TENSOR.TIME_MIX_G2,
        MODEL_TENSOR.TIME_MIX_K_K,
        MODEL_TENSOR.TIME_MIX_K_A,
        MODEL_TENSOR.TIME_MIX_R_K,
        MODEL_TENSOR.TIME_MIX_KEY,
        MODEL_TENSOR.TIME_MIX_VALUE,
        MODEL_TENSOR.TIME_MIX_RECEPTANCE,
        MODEL_TENSOR.TIME_MIX_LN,
        MODEL_TENSOR.TIME_MIX_OUTPUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.MAMBA: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.SSM_IN,
        MODEL_TENSOR.SSM_CONV1D,
        MODEL_TENSOR.SSM_X,
        MODEL_TENSOR.SSM_DT,
        MODEL_TENSOR.SSM_A,
        MODEL_TENSOR.SSM_D,
        MODEL_TENSOR.SSM_OUT,
    ],
    MODEL_ARCH.XVERSE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
```

```
    ],
    MODEL_ARCH.COMMAND_R: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.ATTN_Q_NORM,
    ],
    MODEL_ARCH.COHERE2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.DBRX: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_OUT_NORM,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
    ],
    MODEL_ARCH.OLMO: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.OLMO2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
```

```
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_POST_NORM,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.FFN_POST_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.OLMOE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
    ],
    MODEL_ARCH.OPENELM: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.ARCTIC: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
```

```
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_NORM_EXP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
    ],
    MODEL_ARCH.DEEPSEEK: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
        MODEL_TENSOR.FFN_GATE_SHEXP,
        MODEL_TENSOR.FFN_DOWN_SHEXP,
        MODEL_TENSOR.FFN_UP_SHEXP,
    ],
    MODEL_ARCH.DEEPSEEK2: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_Q_A,
        MODEL_TENSOR.ATTN_Q_B,
        MODEL_TENSOR.ATTN_KV_A_MQA,
        MODEL_TENSOR.ATTN_KV_B,
        MODEL_TENSOR.ATTN_Q_A_NORM,
        MODEL_TENSOR.ATTN_KV_A_NORM,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_GATE_EXP,
```

```
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
        MODEL_TENSOR.FFN_GATE_SHEXP,
        MODEL_TENSOR.FFN_DOWN_SHEXP,
        MODEL_TENSOR.FFN_UP_SHEXP,
        MODEL_TENSOR.FFN_EXP_PROBS_B,
    ],
    MODEL_ARCH.PLM: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_KV_A_MQA,
        MODEL_TENSOR.ATTN_KV_A_NORM,
        MODEL_TENSOR.ATTN_KV_B,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.FFN_DOWN,
    ],
    MODEL_ARCH.CHATGLM : [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.GLM4 : [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.ATTN_POST_NORM,
        MODEL_TENSOR.FFN_POST_NORM,
    ],
    MODEL_ARCH.BITNET: [
```

```
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
        MODEL_TENSOR.ATTN_SUB_NORM,
        MODEL_TENSOR.FFN_SUB_NORM,
    ],
    MODEL_ARCH.T5: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.DEC_ATTN_NORM,
        MODEL_TENSOR.DEC_ATTN_Q,
        MODEL_TENSOR.DEC_ATTN_K,
        MODEL_TENSOR.DEC_ATTN_V,
        MODEL_TENSOR.DEC_ATTN_OUT,
        MODEL_TENSOR.DEC_ATTN_REL_B,
        MODEL_TENSOR.DEC_CROSS_ATTN_NORM,
        MODEL_TENSOR.DEC_CROSS_ATTN_Q,
        MODEL_TENSOR.DEC_CROSS_ATTN_K,
        MODEL_TENSOR.DEC_CROSS_ATTN_V,
        MODEL_TENSOR.DEC_CROSS_ATTN_OUT,
        MODEL_TENSOR.DEC_CROSS_ATTN_REL_B,
        MODEL_TENSOR.DEC_FFN_NORM,
        MODEL_TENSOR.DEC_FFN_GATE,
        MODEL_TENSOR.DEC_FFN_DOWN,
        MODEL_TENSOR.DEC_FFN_UP,
        MODEL_TENSOR.DEC_OUTPUT_NORM,
        MODEL_TENSOR.ENC_ATTN_NORM,
        MODEL_TENSOR.ENC_ATTN_Q,
        MODEL_TENSOR.ENC_ATTN_K,
        MODEL_TENSOR.ENC_ATTN_V,
        MODEL_TENSOR.ENC_ATTN_OUT,
        MODEL_TENSOR.ENC_ATTN_REL_B,
        MODEL_TENSOR.ENC_FFN_NORM,
        MODEL_TENSOR.ENC_FFN_GATE,
        MODEL_TENSOR.ENC_FFN_DOWN,
        MODEL_TENSOR.ENC_FFN_UP,
        MODEL_TENSOR.ENC_OUTPUT_NORM,
    ],
    MODEL_ARCH.T5ENCODER: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ENC_ATTN_NORM,
        MODEL_TENSOR.ENC_ATTN_Q,
        MODEL_TENSOR.ENC_ATTN_K,
        MODEL_TENSOR.ENC_ATTN_V,
        MODEL_TENSOR.ENC_ATTN_OUT,
```

```
        MODEL_TENSOR.ENC_ATTN_REL_B,
        MODEL_TENSOR.ENC_FFN_NORM,
        MODEL_TENSOR.ENC_FFN_GATE,
        MODEL_TENSOR.ENC_FFN_DOWN,
        MODEL_TENSOR.ENC_FFN_UP,
        MODEL_TENSOR.ENC_OUTPUT_NORM,
    ],
    MODEL_ARCH.JAIS: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_QKV,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.NEMOTRON: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.EXAONE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.ATTN_ROT_EMBD,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.GRANITE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
```

```
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.GRANITE_MOE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
    ],
    MODEL_ARCH.CHAMELEON: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_Q_NORM,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_K_NORM,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE,
        MODEL_TENSOR.FFN_DOWN,
        MODEL_TENSOR.FFN_UP,
    ],
    MODEL_ARCH.WAVTOKENIZER_DEC: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.TOKEN_EMBD_NORM,
        MODEL_TENSOR.CONV1D,
        MODEL_TENSOR.CONVNEXT_DW,
        MODEL_TENSOR.CONVNEXT_NORM,
        MODEL_TENSOR.CONVNEXT_PW1,
        MODEL_TENSOR.CONVNEXT_PW2,
        MODEL_TENSOR.CONVNEXT_GAMMA,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.POSNET_CONV1,
        MODEL_TENSOR.POSNET_CONV2,
```

```
        MODEL_TENSOR.POSNET_NORM,
        MODEL_TENSOR.POSNET_NORM1,
        MODEL_TENSOR.POSNET_NORM2,
        MODEL_TENSOR.POSNET_ATTN_NORM,
        MODEL_TENSOR.POSNET_ATTN_Q,
        MODEL_TENSOR.POSNET_ATTN_K,
        MODEL_TENSOR.POSNET_ATTN_V,
        MODEL_TENSOR.POSNET_ATTN_OUT,
    ],
    MODEL_ARCH.BAILINGMOE: [
        MODEL_TENSOR.TOKEN_EMBD,
        MODEL_TENSOR.OUTPUT_NORM,
        MODEL_TENSOR.OUTPUT,
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_NORM,
        MODEL_TENSOR.ATTN_Q,
        MODEL_TENSOR.ATTN_K,
        MODEL_TENSOR.ATTN_V,
        MODEL_TENSOR.ATTN_OUT,
        MODEL_TENSOR.FFN_GATE_INP,
        MODEL_TENSOR.FFN_NORM,
        MODEL_TENSOR.FFN_GATE_EXP,
        MODEL_TENSOR.FFN_DOWN_EXP,
        MODEL_TENSOR.FFN_UP_EXP,
        MODEL_TENSOR.FFN_GATE_SHEXP,
        MODEL_TENSOR.FFN_DOWN_SHEXP,
        MODEL_TENSOR.FFN_UP_SHEXP,
    ],
    # TODO
}

# tensors that will not be serialized
MODEL_TENSOR_SKIP: dict[MODEL_ARCH, list[MODEL_TENSOR]] = {
    MODEL_ARCH.LLAMA: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.DECI: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.BAICHUAN: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.QWEN: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.CODESHELL: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.ORION: [
```

```
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.STARCODER2: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.XVERSE: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.DEEPSEEK: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.DEEPSEEK2: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.CHATGLM: [
        MODEL_TENSOR.ROPE_FREQS,
    ],
    MODEL_ARCH.NEMOTRON: [
        MODEL_TENSOR.ROPE_FREQS,
        MODEL_TENSOR.ATTN_ROT_EMBD,
    ],
    MODEL_ARCH.BAILINGMOE: [
        MODEL_TENSOR.ROPE_FREQS,
    ],
}

#
# types
#


class TokenType(IntEnum):
    NORMAL       = 1
    UNKNOWN      = 2
    CONTROL      = 3
    USER_DEFINED = 4
    UNUSED       = 5
    BYTE         = 6


class RopeScalingType(Enum):
    NONE     = 'none'
    LINEAR   = 'linear'
    YARN     = 'yarn'
    LONGROPE = 'longrope'


class PoolingType(IntEnum):
    NONE = 0
```

```
    MEAN = 1
    CLS  = 2


class GGMLQuantizationType(IntEnum):
    F32     = 0
    F16     = 1
    Q4_0    = 2
    Q4_1    = 3
    Q5_0    = 6
    Q5_1    = 7
    Q8_0    = 8
    Q8_1    = 9
    Q2_K    = 10
    Q3_K    = 11
    Q4_K    = 12
    Q5_K    = 13
    Q6_K    = 14
    Q8_K    = 15
    IQ2_XXS = 16
    IQ2_XS  = 17
    IQ3_XXS = 18
    IQ1_S   = 19
    IQ4_NL  = 20
    IQ3_S   = 21
    IQ2_S   = 22
    IQ4_XS  = 23
    I8      = 24
    I16     = 25
    I32     = 26
    I64     = 27
    F64     = 28
    IQ1_M   = 29
    BF16    = 30
    TQ1_0   = 34
    TQ2_0   = 35


class ExpertGatingFuncType(IntEnum):
    SOFTMAX  = 1
    SIGMOID  = 2


# TODO: add GGMLFileType from ggml_ftype in ggml.h


# from llama_ftype in llama.h
# ALL VALUES SHOULD BE THE SAME HERE AS THEY ARE OVER THERE.
class LlamaFileType(IntEnum):
    ALL_F32              = 0
    MOSTLY_F16           = 1   # except 1d tensors
    MOSTLY_Q4_0          = 2   # except 1d tensors
    MOSTLY_Q4_1          = 3   # except 1d tensors
    # MOSTLY_Q4_1_SOME_F16 = 4   # tok_embeddings.weight and output.weight are F16
```

```
    # MOSTLY_Q4_2        = 5   # support has been removed
    # MOSTLY_Q4_3        = 6   # support has been removed
    MOSTLY_Q8_0          = 7   # except 1d tensors
    MOSTLY_Q5_0          = 8   # except 1d tensors
    MOSTLY_Q5_1          = 9   # except 1d tensors
    MOSTLY_Q2_K          = 10  # except 1d tensors
    MOSTLY_Q3_K_S        = 11  # except 1d tensors
    MOSTLY_Q3_K_M        = 12  # except 1d tensors
    MOSTLY_Q3_K_L        = 13  # except 1d tensors
    MOSTLY_Q4_K_S        = 14  # except 1d tensors
    MOSTLY_Q4_K_M        = 15  # except 1d tensors
    MOSTLY_Q5_K_S        = 16  # except 1d tensors
    MOSTLY_Q5_K_M        = 17  # except 1d tensors
    MOSTLY_Q6_K          = 18  # except 1d tensors
    MOSTLY_IQ2_XXS       = 19  # except 1d tensors
    MOSTLY_IQ2_XS        = 20  # except 1d tensors
    MOSTLY_Q2_K_S        = 21  # except 1d tensors
    MOSTLY_IQ3_XS        = 22  # except 1d tensors
    MOSTLY_IQ3_XXS       = 23  # except 1d tensors
    MOSTLY_IQ1_S         = 24  # except 1d tensors
    MOSTLY_IQ4_NL        = 25  # except 1d tensors
    MOSTLY_IQ3_S         = 26  # except 1d tensors
    MOSTLY_IQ3_M         = 27  # except 1d tensors
    MOSTLY_IQ2_S         = 28  # except 1d tensors
    MOSTLY_IQ2_M         = 29  # except 1d tensors
    MOSTLY_IQ4_XS        = 30  # except 1d tensors
    MOSTLY_IQ1_M         = 31  # except 1d tensors
    MOSTLY_BF16          = 32  # except 1d tensors
    # MOSTLY_Q4_0_4_4      = 33  # removed from gguf files, use Q4_0 and runtime repack
    # MOSTLY_Q4_0_4_8      = 34  # removed from gguf files, use Q4_0 and runtime repack
    # MOSTLY_Q4_0_8_8      = 35  # removed from gguf files, use Q4_0 and runtime repack
    MOSTLY_TQ1_0         = 36  # except 1d tensors
    MOSTLY_TQ2_0         = 37  # except 1d tensors

    GUESSED              = 1024  # not specified in the model file


class GGUFEndian(IntEnum):
    LITTLE = 0
    BIG = 1


class GGUFValueType(IntEnum):
    UINT8   = 0
    INT8    = 1
    UINT16  = 2
    INT16   = 3
    UINT32  = 4
    INT32   = 5
    FLOAT32 = 6
    BOOL    = 7
    STRING  = 8
    ARRAY   = 9
    UINT64  = 10
```

```
    INT64   = 11
    FLOAT64 = 12

    @staticmethod
    def get_type(val: Any) -> GGUFValueType:
        if isinstance(val, (str, bytes, bytearray)):
            return GGUFValueType.STRING
        elif isinstance(val, list):
            return GGUFValueType.ARRAY
        elif isinstance(val, float):
            return GGUFValueType.FLOAT32
        elif isinstance(val, bool):
            return GGUFValueType.BOOL
        elif isinstance(val, int):
            return GGUFValueType.INT32
        # TODO: need help with 64-bit types in Python
        else:
            raise ValueError(f"Unknown type: {type(val)}")


# Items here are (block size, type size)
QK_K = 256
GGML_QUANT_SIZES: dict[GGMLQuantizationType, tuple[int, int]] = {
    GGMLQuantizationType.F32:     (1, 4),
    GGMLQuantizationType.F16:     (1, 2),
    GGMLQuantizationType.Q4_0:    (32, 2 + 16),
    GGMLQuantizationType.Q4_1:    (32, 2 + 2 + 16),
    GGMLQuantizationType.Q5_0:    (32, 2 + 4 + 16),
    GGMLQuantizationType.Q5_1:    (32, 2 + 2 + 4 + 16),
    GGMLQuantizationType.Q8_0:    (32, 2 + 32),
    GGMLQuantizationType.Q8_1:    (32, 4 + 4 + 32),
    GGMLQuantizationType.Q2_K:    (256, 2 + 2 + QK_K // 16 + QK_K // 4),
    GGMLQuantizationType.Q3_K:    (256, 2 + QK_K // 4 + QK_K // 8 + 12),
    GGMLQuantizationType.Q4_K:    (256, 2 + 2 + QK_K // 2 + 12),
    GGMLQuantizationType.Q5_K:    (256, 2 + 2 + QK_K // 2 + QK_K // 8 + 12),
    GGMLQuantizationType.Q6_K:    (256, 2 + QK_K // 2 + QK_K // 4 + QK_K // 16),
    GGMLQuantizationType.Q8_K:    (256, 4 + QK_K + QK_K // 8),
    GGMLQuantizationType.IQ2_XXS: (256, 2 + QK_K // 4),
    GGMLQuantizationType.IQ2_XS:  (256, 2 + QK_K // 4 + QK_K // 32),
    GGMLQuantizationType.IQ3_XXS: (256, 2 + QK_K // 4 + QK_K // 8),
    GGMLQuantizationType.IQ1_S:   (256, 2 + QK_K // 8 + QK_K // 16),
    GGMLQuantizationType.IQ4_NL:  (32, 2 + 16),
    GGMLQuantizationType.IQ3_S:   (256, 2 + QK_K // 4 + QK_K // 8 + QK_K // 32 + 4),
    GGMLQuantizationType.IQ2_S:   (256, 2 + QK_K // 4 + QK_K // 16),
    GGMLQuantizationType.IQ4_XS:  (256, 2 + 2 + QK_K // 2 + QK_K // 64),
    GGMLQuantizationType.I8:      (1, 1),
    GGMLQuantizationType.I16:     (1, 2),
    GGMLQuantizationType.I32:     (1, 4),
    GGMLQuantizationType.I64:     (1, 8),
    GGMLQuantizationType.F64:     (1, 8),
    GGMLQuantizationType.IQ1_M:   (256, QK_K // 8 + QK_K // 16  + QK_K // 32),
    GGMLQuantizationType.BF16:    (1, 2),
    GGMLQuantizationType.TQ1_0:   (256, 2 + 4 * 13),
    GGMLQuantizationType.TQ2_0:   (256, 2 + 64),
```

```
}


# Aliases for backward compatibility.

# general
KEY_GENERAL_ARCHITECTURE         = Keys.General.ARCHITECTURE
KEY_GENERAL_QUANTIZATION_VERSION = Keys.General.QUANTIZATION_VERSION
KEY_GENERAL_ALIGNMENT            = Keys.General.ALIGNMENT
KEY_GENERAL_NAME                 = Keys.General.NAME
KEY_GENERAL_AUTHOR               = Keys.General.AUTHOR
KEY_GENERAL_URL                  = Keys.General.URL
KEY_GENERAL_DESCRIPTION          = Keys.General.DESCRIPTION
KEY_GENERAL_LICENSE              = Keys.General.LICENSE
KEY_GENERAL_SOURCE_URL           = Keys.General.SOURCE_URL
KEY_GENERAL_FILE_TYPE            = Keys.General.FILE_TYPE

# LLM
KEY_VOCAB_SIZE            = Keys.LLM.VOCAB_SIZE
KEY_CONTEXT_LENGTH        = Keys.LLM.CONTEXT_LENGTH
KEY_EMBEDDING_LENGTH      = Keys.LLM.EMBEDDING_LENGTH
KEY_BLOCK_COUNT           = Keys.LLM.BLOCK_COUNT
KEY_FEED_FORWARD_LENGTH   = Keys.LLM.FEED_FORWARD_LENGTH
KEY_USE_PARALLEL_RESIDUAL = Keys.LLM.USE_PARALLEL_RESIDUAL
KEY_TENSOR_DATA_LAYOUT    = Keys.LLM.TENSOR_DATA_LAYOUT

# attention
KEY_ATTENTION_HEAD_COUNT        = Keys.Attention.HEAD_COUNT
KEY_ATTENTION_HEAD_COUNT_KV     = Keys.Attention.HEAD_COUNT_KV
KEY_ATTENTION_MAX_ALIBI_BIAS    = Keys.Attention.MAX_ALIBI_BIAS
KEY_ATTENTION_CLAMP_KQV         = Keys.Attention.CLAMP_KQV
KEY_ATTENTION_LAYERNORM_EPS     = Keys.Attention.LAYERNORM_EPS
KEY_ATTENTION_LAYERNORM_RMS_EPS = Keys.Attention.LAYERNORM_RMS_EPS

# RoPE
KEY_ROPE_DIMENSION_COUNT      = Keys.Rope.DIMENSION_COUNT
KEY_ROPE_FREQ_BASE            = Keys.Rope.FREQ_BASE
KEY_ROPE_SCALING_TYPE         = Keys.Rope.SCALING_TYPE
KEY_ROPE_SCALING_FACTOR       = Keys.Rope.SCALING_FACTOR
KEY_ROPE_SCALING_ORIG_CTX_LEN = Keys.Rope.SCALING_ORIG_CTX_LEN
KEY_ROPE_SCALING_FINETUNED    = Keys.Rope.SCALING_FINETUNED

# SSM
KEY_SSM_CONV_KERNEL    = Keys.SSM.CONV_KERNEL
KEY_SSM_INNER_SIZE     = Keys.SSM.INNER_SIZE
KEY_SSM_STATE_SIZE     = Keys.SSM.STATE_SIZE
KEY_SSM_TIME_STEP_RANK = Keys.SSM.TIME_STEP_RANK
KEY_SSM_DT_B_C_RMS     = Keys.SSM.DT_B_C_RMS

# tokenization
KEY_TOKENIZER_MODEL      = Keys.Tokenizer.MODEL
KEY_TOKENIZER_PRE        = Keys.Tokenizer.PRE
KEY_TOKENIZER_LIST       = Keys.Tokenizer.LIST
KEY_TOKENIZER_TOKEN_TYPE = Keys.Tokenizer.TOKEN_TYPE
```

```
KEY_TOKENIZER_SCORES     = Keys.Tokenizer.SCORES
KEY_TOKENIZER_MERGES     = Keys.Tokenizer.MERGES
KEY_TOKENIZER_BOS_ID     = Keys.Tokenizer.BOS_ID
KEY_TOKENIZER_EOS_ID     = Keys.Tokenizer.EOS_ID
KEY_TOKENIZER_EOT_ID     = Keys.Tokenizer.EOT_ID
KEY_TOKENIZER_EOM_ID     = Keys.Tokenizer.EOM_ID
KEY_TOKENIZER_UNK_ID     = Keys.Tokenizer.UNK_ID
KEY_TOKENIZER_SEP_ID     = Keys.Tokenizer.SEP_ID
KEY_TOKENIZER_PAD_ID     = Keys.Tokenizer.PAD_ID
KEY_TOKENIZER_MASK_ID    = Keys.Tokenizer.MASK_ID
KEY_TOKENIZER_HF_JSON    = Keys.Tokenizer.HF_JSON
KEY_TOKENIZER_RWKV       = Keys.Tokenizer.RWKV

KEY_TOKENIZER_FIM_PRE_ID = Keys.Tokenizer.FIM_PRE_ID
KEY_TOKENIZER_FIM_SUF_ID = Keys.Tokenizer.FIM_SUF_ID
KEY_TOKENIZER_FIM_MID_ID = Keys.Tokenizer.FIM_MID_ID
KEY_TOKENIZER_FIM_PAD_ID = Keys.Tokenizer.FIM_PAD_ID
KEY_TOKENIZER_FIM_REP_ID = Keys.Tokenizer.FIM_REP_ID
KEY_TOKENIZER_FIM_SEP_ID = Keys.Tokenizer.FIM_SEP_ID

# deprecated
KEY_TOKENIZER_PREFIX_ID  = Keys.Tokenizer.PREFIX_ID
KEY_TOKENIZER_SUFFIX_ID  = Keys.Tokenizer.SUFFIX_ID
KEY_TOKENIZER_MIDDLE_ID  = Keys.Tokenizer.MIDDLE_ID

==== context_activator.py ====
import yaml
from pathlib import Path

FRAGMENTS_DIR = Path("fragments/core")
ACTIVATION_LOG = Path("logs/context_activation.log")
ACTIVATION_LOG.parent.mkdir(parents=True, exist_ok=True)

class ContextActivator:
    def __init__(self, activation_threshold=0.75):
        self.threshold = activation_threshold

    def scan_fragments(self):
        activated = []
        for frag_file in FRAGMENTS_DIR.glob("*.yaml"):
            try:
                with open(frag_file, 'r') as f:
                    frag = yaml.safe_load(f)
                if frag.get("confidence", 0.5) >= self.threshold:
                    activated.append(frag)
            except Exception as e:
                print(f"Error reading {frag_file.name}: {e}")
        return activated

    def log_activations(self, activations):
        with open(ACTIVATION_LOG, 'a') as log:
            for frag in activations:
                log.write(f"[ACTIVATED] {frag['id']} :: {frag.get('claim', '???')}\n")
        print(f"[ContextActivator] {len(activations)} fragment(s) activated.")
```

```
    def run(self):
        active = self.scan_fragments()
        self.log_activations(active)

if __name__ == "__main__":
    ctx = ContextActivator()
    ctx.run()

==== convert_hf_to_gguf.py ====
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from __future__ import annotations

import ast
import logging
import argparse
import contextlib
import json
import os
import re
import sys
from enum import IntEnum
from pathlib import Path
from hashlib import sha256
from typing import TYPE_CHECKING, Any, Callable, ContextManager, Iterable, Iterator, Literal, Sequence,
TypeVar, cast
from itertools import chain

import math
import numpy as np
import torch

if TYPE_CHECKING:
    from torch import Tensor

if 'NO_LOCAL_GGUF' not in os.environ:
    sys.path.insert(1, str(Path(__file__).parent / 'gguf-py'))
import gguf

logger = logging.getLogger("hf-to-gguf")


###### MODEL DEFINITIONS ######

class SentencePieceTokenTypes(IntEnum):
    NORMAL = 1
    UNKNOWN = 2
    CONTROL = 3
    USER_DEFINED = 4
    UNUSED = 5
    BYTE = 6
```

```
AnyModel = TypeVar("AnyModel", bound="type[Model]")


class Model:
    _model_classes: dict[str, type[Model]] = {}

    dir_model: Path
    ftype: gguf.LlamaFileType
    fname_out: Path
    is_big_endian: bool
    endianess: gguf.GGUFEndian
    use_temp_file: bool
    lazy: bool
    part_names: list[str]
    is_safetensors: bool
    hparams: dict[str, Any]
    block_count: int
    tensor_map: gguf.TensorNameMap
    tensor_names: set[str] | None
    gguf_writer: gguf.GGUFWriter
    model_name: str | None
    metadata_override: Path | None
    dir_model_card: Path
    remote_hf_model_id: str | None

    # subclasses should define this!
    model_arch: gguf.MODEL_ARCH

      def __init__(self, dir_model: Path, ftype: gguf.LlamaFileType, fname_out: Path, is_big_endian: bool =
False,
                 use_temp_file: bool = False, eager: bool = False,
                 metadata_override: Path | None = None, model_name: str | None = None,
                 split_max_tensors: int = 0, split_max_size: int = 0, dry_run: bool = False,
                   small_first_shard: bool = False, hparams: dict[str, Any] | None = None, remote_hf_model_id:
str | None = None):
        if type(self) is Model:
            raise TypeError(f"{type(self).__name__!r} should not be directly instantiated")

        self.dir_model = dir_model
        self.ftype = ftype
        self.fname_out = fname_out
        self.is_big_endian = is_big_endian
        self.endianess = gguf.GGUFEndian.BIG if is_big_endian else gguf.GGUFEndian.LITTLE
        self.use_temp_file = use_temp_file
        self.lazy = not eager or (remote_hf_model_id is not None)
        self.remote_hf_model_id = remote_hf_model_id
        if remote_hf_model_id is not None:
            self.is_safetensors = True

            def get_remote_tensors() -> Iterator[tuple[str, Tensor]]:
                logger.info(f"Using remote model with HuggingFace id: {remote_hf_model_id}")
                remote_tensors = gguf.utility.SafetensorRemote.get_list_tensors_hf_model(remote_hf_model_id)
                self.tensor_names = set(name for name in remote_tensors.keys())
```