Add a new model architecture to `llama.cpp`（向llama.cpp项目中添加一个新的model architecture）

Adding a model requires few steps（添加一个model需要几个步骤）:

1. Convert the model to GGUF（1、首先需要将模型文件转化为GGUF文件）
2. Define the model architecture in `llama.cpp`（在llama.cpp项目中定义模型架构）
3. Build the GGML graph implementation（构建GGMLgraph implementation）

After following these steps, you can open PR.(完成了上述这些步骤，你可以进行PR)

Also, it is important to check that the examples and main ggml backends (CUDA, METAL, CPU) are working with the new architecture, especially（另外，检查示例和主要 ggml 后端（CUDA、METAL、CPU）是否与新架构兼容也很重要，尤其是）:

- main
- imatrix
- quantize
- server

1. Convert the model to GGUF（第一步：将model转化成GGUF）

This step is done in python with a `convert` script using the gguf library. Depending on the model architecture, you can use either convert_hf_to_gguf.py or examples/convert_legacy_llama.py (for `llama/llama2` models in `.pth` format).

- 此步骤在 Python 中使用 gguf 库的 `convert` 脚本完成。
- 根据模型架构，您可以使用 convert_hf_to_gguf.py 或 examples/convert_legacy_llama.py（用于 `.pth` 格式的 `llama/llama2` 模型）。

The convert script reads the model configuration, tokenizer, tensor names+data and converts them to GGUF metadata and tensors.（转换脚本读取模型配置、标记器、张量名称+数据并将它们转换为 GGUF 元数据和张量。）

The required steps to implement for an HF model are（实现 HF 模型所需的步骤如下）:

1. Define the model `Model.register` annotation in a new `Model` subclass, example（在新的 Model 子类中定义模型 Model.register 注释）:

```python
@Model.register("MyModelForCausalLM")
class MyModel(Model):
    model_arch = gguf.MODEL_ARCH.GROK
```

2. Define the layout of the GGUF tensors in constants.py（定义GGUF tensors的布局）

Add an enum entry in `MODEL_ARCH`, the model human friendly name in `MODEL_ARCH_NAMES` and the GGUF tensor names in `MODEL_TENSORS`.（在 `MODEL_ARCH` 中添加枚举条目，在 `MODEL_ARCH_NAMES` 中添加模型人性

化名称，在 `MODEL_TENSORS` 中添加 GGUF 张量名称。）

Example for `falcon` model（falcon模型的例子）:

```
    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,
    ]
```

3. Map the original tensor names to the standardize equivalent in GGUF（将原始张量名称映射到 GGUF 中的标准化等效项）

As a general rule, before adding a new tensor name to GGUF, be sure the equivalent naming does not already exist.（一般来说，在向 GGUF 添加新的张量名称之前，请确保等效命名尚不存在。）

Once you have found the GGUF tensor name equivalent, add it to the tensor_mapping.py file.（一旦你发现GGUF tensor名字是相等的，将其添加到tensor_mapping.py中）

If the tensor name is part of a repetitive layer/block, the key word `bid` substitutes it.（如果tensor name是重复layer/block的一部分，使用关键词"bid"替代了它。）

Example for the normalization tensor in attention layers（注意力层中规范化张量的示例）:

```
block_mappings_cfg: dict[MODEL_TENSOR, tuple[str, ...]] = {
        # Attention norm
        MODEL_TENSOR.ATTN_NORM: (
            "gpt_neox.layers.{bid}.input_layernorm",               # gptneox
            "transformer.h.{bid}.ln_1",                            # gpt2 gpt-j
refact qwen
            "transformer.blocks.{bid}.norm_1",                     # mpt
            ...
        )
}
```

`transformer.blocks.{bid}.norm_1` will be mapped to `blk.{bid}.attn_norm` in GGUF.

Depending on the model configuration, tokenizer, code and tensors layout, you will have to override:

- `Model#set_gguf_parameters`
- `Model#set_vocab`
- `Model#write_tensors`

NOTE: Tensor names must end with `.weight` suffix, that is the convention and several tools like `quantize` expect this to proceed the weights.

2. Define the model architecture in `llama.cpp`（在llama.cpp中定义模型架构）

The model params and tensors layout must be defined in `llama.cpp`（模型的参数、tensors layout必须在llama.cpp中进行定义）:

1. Define a new `llm_arch`
2. Define the tensors layout in `LLM_TENSOR_NAMES`
3. Add any non standard metadata in `llm_load_hparams`
4. Create the tensors for inference in `llm_load_tensors`
5. If the model has a RoPE operation, add the rope type in `llama_rope_type`

NOTE: The dimensions in `ggml` are typically in the reverse order of the `pytorch` dimensions.

3. Build the GGML graph implementation（构建GGML graph implementation）

This is the funniest part, you have to provide the inference graph implementation of the new model architecture in `llama_build_graph`.(这是最有趣的部分，您必须“llama_build_graph”中提供新模型架构的推理图实现。)

Have a look at existing implementation like `build_llama`, `build_dbrx` or `build_bert`.（查看已存在的实现例如build_llama\buid_dbrx\build_bert）

When implementing a new graph, please note that the underlying `ggml` backends might not support them all, support for missing backend operations can be added in another PR.（当实现一个新的图表时，请注意底层的“ggml”后端可能不支持它们全部，对缺失的后端操作的支持可以在另一个 PR 中添加。）

Note: to debug the inference graph: you can use llama-eval-callback.

GGUF specification

https://github.com/ggerganov/ggml/blob/master/docs/gguf.md

Resources

- YaRN RoPE scaling https://github.com/ggerganov/llama.cpp/pull/2268
- support Baichuan serial models https://github.com/ggerganov/llama.cpp/pull/3009
- support attention bias https://github.com/ggerganov/llama.cpp/pull/4283
- Mixtral support https://github.com/ggerganov/llama.cpp/pull/4406
- BERT embeddings https://github.com/ggerganov/llama.cpp/pull/5423
- Grok-1 support https://github.com/ggerganov/llama.cpp/pull/6204
- Command R Plus support https://github.com/ggerganov/llama.cpp/pull/6491
- support arch DBRX https://github.com/ggerganov/llama.cpp/pull/6515
- How to convert HuggingFace model to GGUF format https://github.com/ggerganov/llama.cpp/discussions/2948