```
class LlamaRMSNorm(nn.Module):
                                                                           def __init__(self, hidden_size, eps=1e-6):
                                               nn.Embedding()
                                                                                """LlamaRMSNorm is equivalent to T5LayerNorm"""
                                                                                super().__init__()
                                                                                self.weight = nn.Parameter(torch.ones(hidden_size))
                                                                                self.variance_epsilon = eps
                                               inputs_embeds
                                                                           def forward(self, hidden_states):
                                                                               input_dtype = hidden_states.dtype
                                                `position_ids`
                                                                               hidden_states = hidden_states.to(torch.float32)
                                          and `causal_mask`
                                                                               variance = hidden_states.pow(2).mean(-1, keepdim=True)
                                           and `past_key_values`
                                                                               hidden_states = hidden_states * torch.rsqrt(
                                                                                    variance + self.variance_epsilon)
                                                                                return self.weight * hidden_states.to(input_dtype)
   LlamaModel(LlamaPreTrainedModel). hidden_states = inputs_embeds
   forward(input_ids)
                                                                                                     hidden_states
                                                                                                                        hidden_states
                                       layer_outputs = LlamaDecoderLayer(config, layer_idx)
                                       forward(hidden_states,causal_mask,
                num_hidden_layers
                                                                                                                         LlamaRMSNorm()
                                                                                                     LlamaRMSNorm()
                                    X | position_ids, past_key_values)
                                       hidden_states = layer_outputs[0]
                                                                                                    LlamaAttention()
                                                                                                                           LlamaMLP()
                                               LlamaRMSNorm()
                                                   return
class LlamaMLP(nn.Module):
                                                                                                                        return
   def __init__(self, config):
                                                                                                                        (hidden_states,)
       super().__init__()
        self.config = config
        self.hidden_size = config.hidden_size
        self.intermediate_size = config.intermediate_size
        self.gate_proj = nn.Linear(self.hidden_size, self.intermediate_size, bias=False)
        self.up_proj = nn.Linear(self.hidden_size, self.intermediate_size, bias=False)
        self.down_proj = nn.Linear(self.intermediate_size, self.hidden_size, bias=False)
                                                                                                   FFN_{GLU}(x, W, V, W_2) = (\sigma(xW) \otimes xV)W_2
        self.act_fn = ACT2FN[config.hidden_act]
                                                                                                 FFN_{Bilinear}(x, W, V, W_2) = (xW \otimes xV)W_2
                                  W_{down}(f(W_{gate}x)\otimes W_{up}x)
                                                                                                 FFN_{ReGLU}(x, W, V, W_2) = (\max(0, xW) \otimes xV)W_2
   def forward(self, x):
                                                                                                 FFN_{GEGLU}(x, W, V, W_2) = (GELU(xW) \otimes xV)W_2
        down_proj = self.down_proj(self.act_fn(self.gate_proj(x)) * self.up_proj(x))
                                                                                                 FFN_{SwiGLU}(x, W, V, W_2) = (Swish_1(xW) \otimes xV)W_2
        return down_proj
```