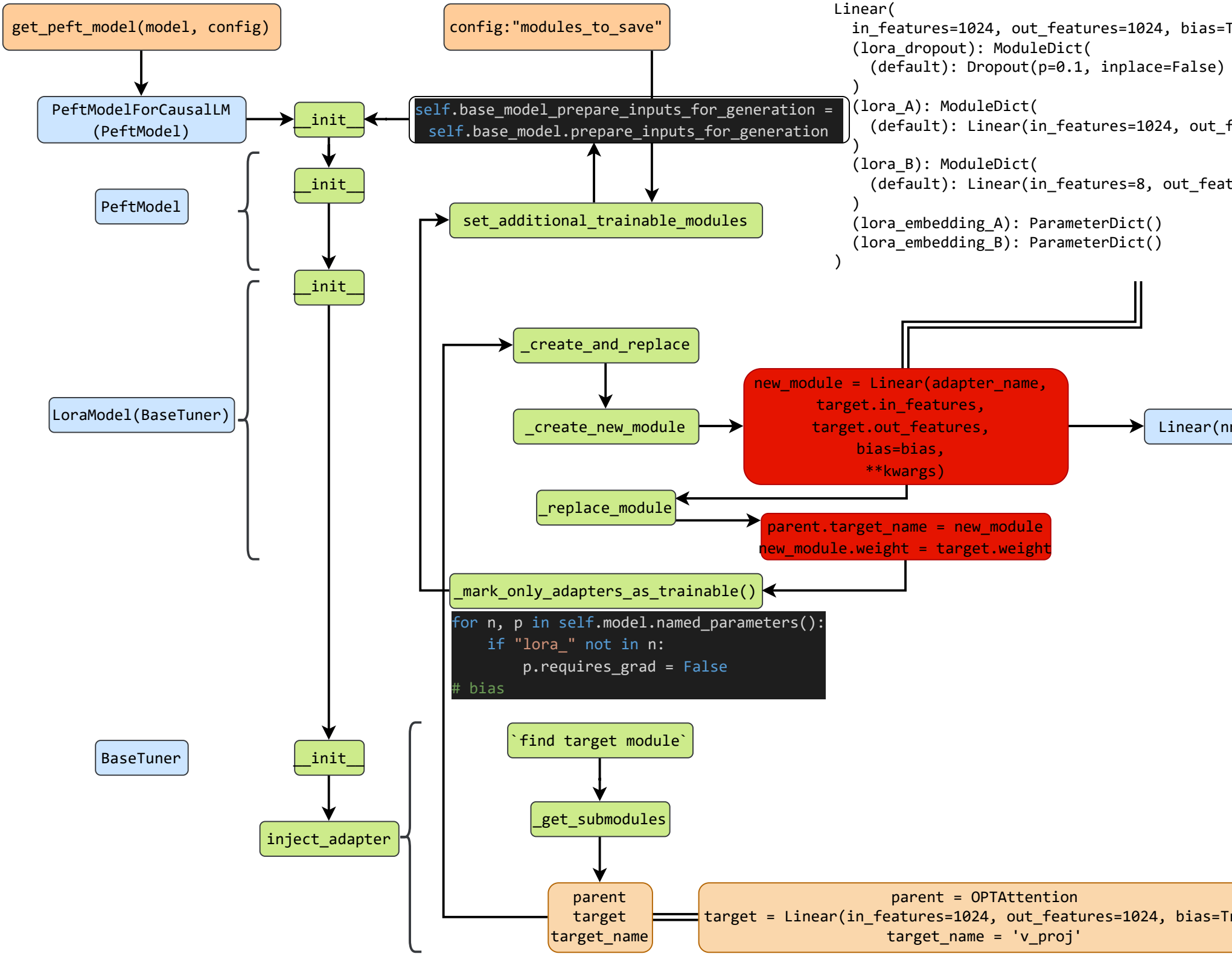
get_peft_model(model, config)
config:"modules_to_save"
Linear(
in_features=1024, out_features=1024, bias=T
(lora_dropout): ModuleDict(
(default): Dropout(p=0.1, inplace=False)
)
(lora_A): ModuleDict(
(default): Linear(in_features=1024, out_f
)
(lora_B): ModuleDict(
(default): Linear(in_features=8, out_feat
)
(lora_embedding_A): ParameterDict()
(lora_embedding_B): ParameterDict()
)
PeftModelForCausalLM
(PeftModel)
__init__
self.base_model_prepare_inputs_for_generation =
self.base_model.prepare_inputs_for_generation
PeftModel
__init__
set_additional_trainable_modules
__init__
LoraModel(BaseTuner)
_create_and_replace
_create_new_module
new_module = Linear(adapter_name,
target.in_features,
target.out_features,
bias=bias,
**kwargs)
Linear(nr
_replace_module
parent.target_name = new_module
new_module.weight = target.weight
_mark_only_adapters_as_trainable()
for n, p in self.model.named_parameters():
if "lora_" not in n:
p.requires_grad = False
# bias
BaseTuner
__init__
`find target module`
_get_submodules
inject_adapter
parent
target
target_name
parent = OPTAttention
target = Linear(in_features=1024, out_features=1024, bias=Tr
target_name = 'v_proj'

rue

features=8, bias=False)

tures=1024, bias=False)

nn.Linear

reset_parameters

```
init.kaiming_uniform_(self.weight, a=math.sqrt(5))
if self.bias is not None:
    fan_in, _ = init._calculate_fan_in_and_fan_out(self.weight)
    bound = 1 / math.sqrt(fan_in) if fan_in > 0 else 0
    init.uniform_(self.bias, -bound, bound)
```

__init__

```
self.in_features
self.out_features
self.weight
[self.bias]
```

n.Linear, LoraLayer)

__init__

```
self.weight.requires_grad = False
if fan_in_fan_out:self.weight.data = self.weight.data.T
nn.Linear.reset_parameters(self)
self.update_layer(adapter_name, r, lora_alpha, lora_dropout, init_lora_weights)
```

LoraLayer

__init__

update_layer

```
if lora_dropout > 0.0:lora_dropout_layer = nn.Dropout(p=lora_dropout)
else:lora_dropout_layer = nn.Identity()
self.lora_dropout.update(nn.ModuleDict({adapter_name: lora_dropout_layer}))
if r > 0:
    self.lora_A.update(nn.ModuleDict({adapter_name: nn.Linear(self.in_features, r, bias=False)}))
    self.lora_B.update(nn.ModuleDict({adapter_name: nn.Linear(r, self.out_features, bias=False)}))
    self.scaling[adapter_name] = lora_alpha / r
if init_lora_weights:
    self.reset_lora_parameters(adapter_name)
```

reset_lora_parameters

```
if adapter_name in self.lora_A.keys():
    # initialize A the same way as the default for nn.Linear and B to zero
    nn.init.kaiming_uniform_(self.lora_A[adapter_name].weight, a=math.sqrt(5))
    nn.init.zeros_(self.lora_B[adapter_name].weight)
if adapter_name in self.lora_embedding_A.keys():
    # initialize a the same way as the default for nn.linear and b to zero
    nn.init.zeros_(self.lora_embedding_A[adapter_name])
    nn.init.normal_(self.lora_embedding_B[adapter_name])
```

rue)