```
import torch
import transformers as tr
```

```
amateur_path = 'Qwen/Qwen2.5-Coder-0.5B-Instruct'
expert_path = 'Qwen/Qwen2.5-3B-Instruct'
```

```
amateur_tokenizer = tr.AutoTokenizer.from_pretrained(amateur_path)
expert_tokenizer = tr.AutoTokenizer.from_pretrained(expert_path)
```

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

```
amateur_model = tr.AutoModelForCausalLM.from_pretrained(amateur_path, torch_dtype=torch.bfloat16).to("cpu")
expert_model = tr.AutoModelForCausalLM.from_pretrained(expert_path, torch_dtype=torch.bfloat16).to("cpu")
```

```
Loading checkpoint shards: 100%                    2/2 [00:01<00:00,  1.17it/s]
generation_config.json: 100%                   242/242 [00:00<00:00, 20.0kB/s]
```

```
def contrastive_decoding(amateur, expert, tokenizer, prompt, max_tokens=100):
    """Implement contrastive decoding using token-level selection."""
    amateur.eval()
    expert.eval()


    input_ids = tokenizer(prompt, return_tensors='pt').input_ids.to(next(expert.parameters()).device)

    output_ids = input_ids.clone()
    amateur_past_key_values = None
    expert_past_key_values = None

    for _ in range(max_tokens):

        with torch.no_grad():
            amateur_out = amateur(input_ids, past_key_values=amateur_past_key_values, use_cache=True)
            expert_out = expert(input_ids, past_key_values=expert_past_key_values, use_cache=True)

        amateur_logits = amateur_out.logits[:, -1, :]
        expert_logits = expert_out.logits[:, -1, :]
```

```
        amateur_past_key_values = amateur_out.past_key_values
        expert_past_key_values = expert_out.past_key_values


        contrastive_logits = expert_logits - amateur_logits
        next_token = torch.argmax(contrastive_logits, dim=-1)


        output_ids = torch.cat([output_ids, next_token.unsqueeze(-1)], dim=-1)
        input_ids = next_token.unsqueeze(-1).to(next(expert.parameters()).device)


        if next_token.item() == tokenizer.eos_token_id:
            break


        if output_ids.shape[1] > max_tokens:
            break

    return tokenizer.decode(output_ids[0], skip_special_tokens=True)


if __name__ == "__main__":
    try:
        user_prompt = "Explain contrastive decoding in simple terms."
        output = contrastive_decoding(amateur_model, expert_model, expert_tokenizer, user_prompt)
        print(output)
    except torch.cuda.OutOfMemoryError:
        print("CUDA ran out of memory. Try reducing the model size or running on CPU.")
    except Exception as e:
        print(f"An error occurred: {e}")
```

```
Explain contrastive decoding in simple terms. Contrast Accountochenlernen hôher consider tín contrast İşteingendencies nämlichemlıcomings Franç:@""Sie”<|fim_pad|> #### Undert
```