Merged LoRA Resume Summarizer & Extraction Report

Summary:

- The base model *mistralai/Mistral-7B-Instruct-v0.2* and two LoRAs for summarization and entity extraction were selected.
- Individual testing showed strong performance in each LoRA's specialized task.
- A linear merge combined both LoRAs equally using `arcee-ai/MergeKit`.
- The merged model successfully handled both summarization and extraction, with slight trade-offs in precision.
- SLERP and TIES merging was attempted but faced parameter-related errors still under resolution.

I. Step 1: Individual Model Testing Before Merge:

- Before merging, both LoRA models were tested individually on the same resume inputs to evaluate their behavior and outputs:

❖ Base Model : mistralai/Mistral-7B-Instruct-v0.2 :

Model Purpose:

The base model *mistralai/Mistral-7B-Instruct-v0.2* was selected due to its strong instruction-tuning, making it highly adaptable for multi-task objectives like summarization and entity extraction.

Supporting Code:

```
from transformers import AutoModelForCausalLM, AutoTokenizer, BitsAndBytesConfig

base_model_id = "mistralai/Mistral-7B-Instruct-v0.2"

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_compute_dtype="float16",
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
)
```

```
tokenizer = AutoTokenizer.from_pretrained(base_model_id)
base_model = AutoModelForCausalLM.from_pretrained(
    base_model_id,
    quantization_config=bnb_config,
    device_map="auto"
)
```

❖ Model-1: waelChafei/resume-summarization-finetuned-mistral-7b

Model Purpose:

This model excels at generating short, coherent professional summaries from resumes.

Supporting Code:

```
from peft import PeftModel
import torch
# Load summarization LoRA
summarizer_model_id = "waelChafei/resume-summarization-finetuned-mistral-7b"
model_summarizer = PeftModel.from_pretrained(base_model, summarizer_model_id)
# Loop for resume text input
while True:
    resume_text = input("\n📝 Paste resume text for summarization (or type 'exit'): ")
    if resume_text.lower() == "exit":
        print("Exiting summarization loop.")
        break
    # prompt structure
    wrapped_prompt = f"""### Instruction:
Summarize the following resume in 2 lines.
```

```
### Input:
{resume_text}
### Response:"""
    inputs = tokenizer(wrapped_prompt, return_tensors="pt").to(model_summarizer.device)
    with torch.no_grad():
        outputs = model_summarizer.generate
        (
            **inputs,
            max_new_tokens=300,
            temperature=0.7,
            top_k=50,
            top_p=0.95,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id,
            eos_token_id=tokenizer.eos_token_id
        )
```

```
full_output = tokenizer.decode(outputs[0], skip_special_tokens=True)

    # Extract clean response
    if "### Response:" in full_output:
        response = full_output.split("### Response:")[-1].strip()
    else:
        response = full_output.strip()

    print("\n📤 Summary:\n", response)
```

Prompt Given:

"Summarize the following resume in 2 lines:

Resume:John Doe is a software engineer with over 5 years of experience at Google working on scalable ML systems. He graduated with a B.Sc in Computer Science from MIT in 2018. His key skills include Python, TensorFlow, distributed systems, and MLOps. He has contributed to several open-source projects and has strong expertise in backend engineering. You can contact him at john.doe@gmail.com."

❖ **OUTPUT:**

```
"Software Engineer at Google with 5 years of experience in scalable ML systems.
Holds a B.Sc in Computer Science from MIT (2018).
Skilled in Python, TensorFlow, distributed systems, and MLOps.
Contributed to open-source projects and proficient in backend engineering.
Contact: john.doe@gmail.com.
```

❖ **FINDINGS:**

☑ Strengths: Coherent and compact summaries.

❌ Weaknesses:

- Does not extract structured fields (e.g., skills, contact, education), If mentioned in the prompt, it echoes the prompt as response.
- Takes input in the format of :
  [### Instruction: ### Input:{resume_text} ### Response:"""]
  Hence requires prompt-wrapping.

❖ Model-2: Pennlaine/Mistral-7B-v02-Entity-Extraction

Model Purpose:
Designed for entity extraction — pulls out structured fields from unstructured resumes.

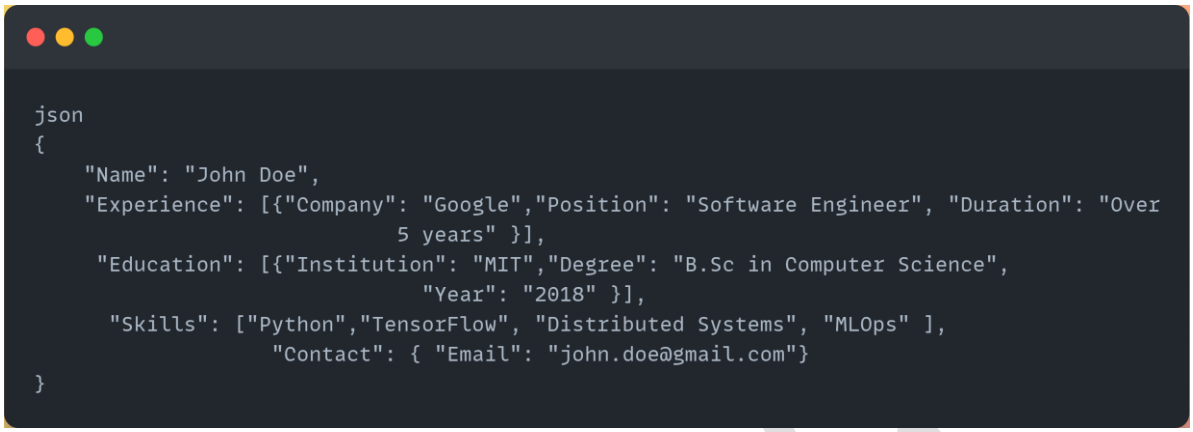
Supporting Code:

```python
# Re-initialize base model
base_model = AutoModelForCausalLM.from_pretrained(
    base_model_id,
    quantization_config=bnb_config,
    device_map="auto"
)

# Load entity extraction LoRA
entity_model_id = "Pennlaine/Mistral-7B-v02-Entity-Extraction"
model_entity = PeftModel.from_pretrained(base_model, entity_model_id)

# Prompt-based test
while True:
    prompt = input("\n📝 Enter resume prompt for entity extraction model (or 'exit'): ")
    if prompt.lower() == "exit":
        break

    inputs = tokenizer(prompt, return_tensors="pt").to(model_entity.device)

    with torch.no_grad():
        outputs = model_entity.generate(
            **inputs,
            max_new_tokens=300,
            temperature=0.7,
            top_k=50,
            top_p=0.95,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id,
            eos_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    print("\n📬 Response:\n", response)
```

❖ Prompt-Given:

Extract key details like -name -experience -education -email from the Resume :John Doe is a software engineer with over 5 years of experience at Google working on scalable ML systems. He graduated with a B.Sc in Computer Science from MIT in 2018. His key skills include Python, TensorFlow, distributed systems, and MLOps. He has contributed to several open-source projects and has strong expertise in backend engineering. You can contact him at john.doe@gmail.com.

❖ **OUTPUT:**



```
json
{
    "Name": "John Doe",
    "Experience": [{"Company": "Google","Position": "Software Engineer", "Duration": "Over
                            5 years" }],
     "Education": [{"Institution": "MIT","Degree": "B.Sc in Computer Science",
                                "Year": "2018" }],
      "Skills": ["Python","TensorFlow", "Distributed Systems", "MLOps" ],
                    "Contact": { "Email": "john.doe@gmail.com"}
}
```

❖ **FINDINGS:**

☑ Strengths: Highly accurate field extraction. Extracts only the mentioned features, doesn't produce inaccurate data or noise.

❌ Weaknesses: No narrative summary generation, if mentioned in the prompt, it echoes the given prompt. Requires wrapping of response as it gives output in **json** format.

II. Step 2: Merging the Two Lora - models:

❖ **Theory:**

Merging LoRA adapters enables parameter-efficient multi-task learning without retraining the base model from scratch. This approach leverages the modularity of LoRA, where each adapter fine-tunes only a subset of parameters. By merging, we effectively combine task-specific adaptations, enabling the model to generalize across both tasks (summarization and entity extraction) while maintaining a lightweight footprint. This approach is particularly beneficial in low-resource deployment scenarios and aligns with the principle of continual learning, where knowledge from separate tasks can be fused without catastrophic forgetting.

❖ **Why Merge?**

Merging the two allowed us to:

- Simplify deployment (1 model instead of 2)
- Reduce inference overhead.
- Handle both tasks (summarization + extraction) in one shot.
- Optimize for memory via 4-bit quantization.

❖ Linear-Merge approach:

The linear merge approach was chosen for its simplicity and effectiveness, allowing both LoRA models to contribute equally by averaging their weights without introducing interpolation complexities, ensuring a balanced output between summarization and entity extraction tasks. The linear merge operates under the assumption that both LoRA adapters contribute equally and independently to the base model's latent space. This is theoretically supported by **linear mode connectivity** in neural networks, which posits that models fine-tuned from a common base can often be interpolated linearly in weight space without performance collapse. As such, linear merging becomes a reliable first approach when tasks are non-conflicting or complementary.

❖ YML File Configuration:

The following yml configuration was stored in a file labelled "***linear.yml***" with path **/contents/linear.yml** :

linear.yml configuration

```yaml
models:
  - model: mistralai/Mistral-7B-Instruct-v0.2
    lora:
      path: waelChafei/resume-summarization-finetuned-mistral-7b
    parameters:
      weight: 1.0

  - model: mistralai/Mistral-7B-Instruct-v0.2
    lora:
      path: Pennlaine/Mistral-7B-v02-Entity-Extraction
    parameters:
      weight: 1.0

merge_method: linear
tokenizer_source: mistralai/Mistral-7B-Instruct-v0.2
dtype: float16
```

❖ Mergekit:

MergeKit simplifies the practical application of model merging, but the technique draws from the broader field of modular transfer learning. This paradigm supports combining independently fine-tuned components (e.g., LoRA adapters) for downstream efficiency and adaptability. This is conceptually related to mixture-of-experts (MoE) architectures, though MoE activates experts dynamically while merging statically combines them.

❖ Arcee-AI Mergekit:

Arcee-AI's MergeKit was used to efficiently merge the base and LoRA models because of its flexibility, ease of use, and support for various merge strategies like linear and slerp. It was installed via pip using:

- *!pip install git+https://github.com/arcee-ai/mergekit.git*

❖ MergeKit Setup and Execution:

1. Install dependencies:

```
# Install dependencies
!pip install git+https://github.com/arcee-ai/mergekit.git
!pip install torch pyyaml huggingface_hub

from huggingface_hub import login
login("hugging-face-access-token")
```

2. Perform Merge after parsing "Linear.yml" File:

```
import torch
import yaml
from mergekit.config import MergeConfiguration
from mergekit.merge import MergeOptions, run_merge

CONFIG_YML = "/content/linear.yml"

with open(CONFIG_YML, "r", encoding="utf-8") as fp:
    merge_config = MergeConfiguration.model_validate(yaml.safe_load(fp))

run_merge(
    merge_config,
    out_path="./models/mergedMistral2",
    options=MergeOptions(
        lora_merge_cache="/tmp",
        cuda=torch.cuda.is_available(),
        copy_tokenizer=True,
        lazy_unpickle=False,
        low_cpu_memory=False,
    ),
)

print(" Merge complete!")
```

3. Upload to Hugging face to load the model for future applications:

After successful merging, the model was uploaded to the Hugging Face Model Hub.

```
from huggingface_hub import create_repo
create_repo("merged-mistral-lora2", private=False)

from huggingface_hub import HfApi, HfFolder, upload_folder
upload_folder(
    folder_path="./models/mergedMistralslerp",  # local path
    repo_id="Impulse007/merged-mistral-lora2",  # your HF repo
    repo_type="model",
)
```

III. Step 3: Testing The Merged Model:

❖ Load with 4-bit Quantization for Efficient Inference:

To reduce memory usage and run on consumer GPUs (or even CPU), we used BitsAndBytes for **4-bit inference** with nf4 quantization:

- Reduces model size significantly
- Maintains performance
- Speeds up inference

❖ Installation & Model Loading:

```
from transformers import AutoTokenizer, AutoModelForCausalLM, BitsAndBytesConfig, pipeline
import torch

model_id = "Impulse007/merged-mistral-lora2"

quant_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)

tokenizer = AutoTokenizer.from_pretrained(model_id)
model = AutoModelForCausalLM.from_pretrained(
    model_id,
    device_map="auto",
    quantization_config=quant_config
)

pipe = pipeline("text-generation", model=model, tokenizer=tokenizer)
```

❖ Resume Prompt Formatting & Interactive Testing:

To standardize interactions across use cases, we implemented a format_prompt() utility supporting 3 modes:

1. Prompt Format Modes:

```
def format_prompt(text, mode="both"):
    text = text.strip()
    if mode == "summarize":
        return f"Summarize the following resume. Only provide a short professional summary.\n\nResume:\n{text}"
```

```
elif mode == "extract":
        return (
            "Extract structured information from the following resume. "
            "Format your response with key fields like:\n"
            "- Name\n- Role\n- Experience\n- Education\n- Skills\n- Contact\n\nResume:\n" + text
        )
    elif mode == "both":
        return (
            "Summarize and extract key information from the following resume. "
            "First, provide a short professional summary. Then list extracted fields clearly as:\n"
            "- Name\n- Role\n- Experience\n- Education\n- Skills\n- Contact\n\nResume:\n" + text
        )
    else:
        return f"Summarize and extract details:\n\n{text}"
```

2. Interactive Testing:

```
def process_resumes_interactively():
    print("💡 Enter 'exit' anytime to stop.\n")

    while True:
        # Select mode
        mode = input("🔧 Enter mode (summarize / extract / both): ").strip().lower()
        if mode == "exit":
            break
        if mode not in ["summarize", "extract", "both"]:
            print("❗ Invalid mode. Choose from summarize / extract / both.")
            continue
# Input resume text
        resume_text = input("📝 Paste the resume text: ").strip()
        if resume_text.lower() == "exit":
            break

        # Format and run
        prompt = format_prompt(resume_text, mode=mode)
        output = pipe(prompt, max_new_tokens=300, do_sample=True, temperature=0.7)[0]["generated_text"]

        # Display result
        print("\n📬 Output:\n" + "-"*50)
        print(output)
        print("-"*50 + "\n")
process_resumes_interactively()
```

❖ **Prompts given with responses:**

1. **Summarization:**

Enter mode (summarize / extract / both): summarize
Paste the resume text: John Doe is a software engineer with over 5 years of experience at Google working on scalable ML systems. He graduated with a B.Sc in Computer Science from MIT in 2018. His key skills include Python, TensorFlow, distributed systems, and MLOps. He has contributed to several open-source projects and has strong expertise in backend engineering. You can contact him at john.doe@gmail.com.

Response:

```
Experienced Software Engineer with a B.Sc in Computer Science from MIT and over 5 years of experience at Google in ML systems development.
Skilled in Python, TensorFlow, distributed systems, and MLOps.
Strong background in backend engineering with contributions to open-source projects
```

2. **Entity-Extraction:**

Enter mode (summarize / extract / both): extract
Paste the resume text: John Doe is a software engineer with over 5 years of experience at Google working on scalable ML systems. He graduated with a B.Sc in Computer Science from MIT in 2018. His key skills include Python, TensorFlow, distributed systems, and MLOps. He has contributed to several open-source projects and has strong expertise in backend engineering. You can contact him at john.doe@gmail.com.

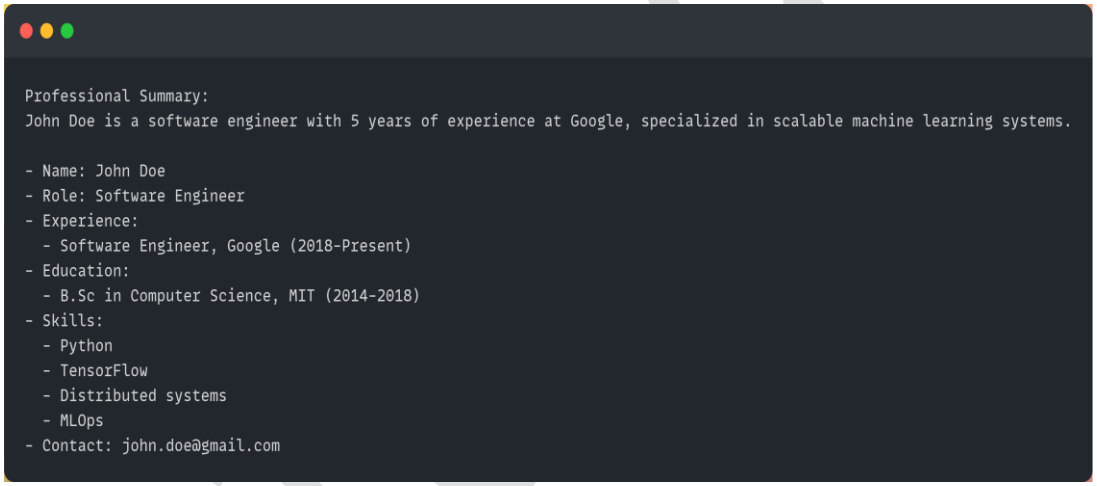
Response:

```
Name: John Doe
Role: Software Engineer
Experience:
- Software Engineer, Google (2018-present)
Education:
- B.Sc in Computer Science, Massachusetts Institute of Technology (MIT) (2014-2018)
Skills:
- Python
- TensorFlow
- Distributed systems
- MLOps
Contact: john.doe@gmail.com
```

3. **Both Summarization and Entity-Extraction:**

Enter mode (summarize / extract / both): both
Paste the resume text: John Doe is a software engineer with over 5 years of experience at Google working on scalable ML systems. He graduated with a B.Sc in Computer Science from MIT in 2018. His key skills include Python, TensorFlow, distributed systems, and MLOps. He has contributed to several open-source projects and has strong expertise in backend engineering. You can contact him at john.doe@gmail.com.

Response:



```
Professional Summary:
John Doe is a software engineer with 5 years of experience at Google, specialized in scalable machine learning systems.

- Name: John Doe
- Role: Software Engineer
- Experience:
  - Software Engineer, Google (2018-Present)
- Education:
  - B.Sc in Computer Science, MIT (2014-2018)
- Skills:
  - Python
  - TensorFlow
  - Distributed systems
  - MLOps
- Contact: john.doe@gmail.com
```

❖ FINAL-FINDINGS:

Theoretical-Implications:
The success of the merged model validates the hypothesis that LoRA adapters encode task-specific capabilities in a modular fashion. When merged, they allow a single model instance to inherit multi-task competencies without retraining. This approach opens pathways for scalable NLP systems where numerous skills can be modularly trained and merged, akin to **plug-and-play AI systems**.

☑ **Strengths:**

- The merged model accurately summarized resumes into short, professional summaries.
- It extracted structured fields like Name, Role, Skills, and Contact with high consistency.
- Lightweight 4-bit quantized loading made the model fast and memory-efficient on standard GPUs.
- The linear merge preserved both LoRA adaptations, enabling combined summarization and entity extraction effectively.

- The model handled both short and detailed resumes without significant hallucination or format errors.

❌ **Weaknesses:**

- Some minor overlap between summarization and extraction fields was observed in "both" mode outputs.
- The model slightly favoured summarization style over strict extraction when prompts were ambiguous.
- Performance might drop if resumes have unusual layouts or missing conventional fields (e.g., no clear Education section).
- Fine-grained customization (like weight adjustment between LoRAs) was limited under a simple linear merge.
- The model still requires manual formatting corrections in some rare edge cases for enterprise-grade applications.

❖ Summary of Achievements:

- ✓ Successfully merged two task-specific LoRA adapters into a single unified Mistral 7B model
- ✓ Used 4-bit quantization to optimize for memory and performance
- ✓ Uploaded the merged model to Hugging Face
- ✓ Developed an interactive CLI to evaluate summarization and information extraction from resumes