Database processing:

- **Format specific handlers for pdf, txt, latex, etc.**

1. Data Preparation (Kaggle-Friendly)

- Semantic Chunking
  - *Kaggle tip*: Pre-process data offline (e.g., Google Colab) and upload as cleaned CSV to Kaggle Datasets.
- Noise Removal
  - **Regex to strip citations (`[1]`, `(Author et al., 2023)`), headers/footers, and equations.**
  - Keep only prose (full sentences/paragraphs).
- Add Diversity
  - **Mix in textbook excerpts (e.g., upload `wikipedia.csv` alongside papers) for broader context.**

2. Instruction Tuning (Pre-Training Phase)

- Q&A Pair Generation
  - Automate conversion of paper sections to Q&A pairs:
    - python
    - Copy
    - Download

```
# Example: Turn "Methodology" into Q&A
questions = ["How does the paper implement X?", "What methodology was used for Y?"]
    answers = [text_chunk_1, text_chunk_2]  # Direct from paper
```

  - Use `datasets` library to create a HuggingFace Dataset for training.
- Prompt Engineering
  - Prefix prompts during training:
    - python
    - Copy
    - Download

```
prompt_template = "Answer concisely and accurately: {question}\nAnswer: {answer}"
```

  - *Kaggle tip*: Use `transformers.Trainer` with `data_collator` for dynamic padding.

3. Model & Tokenizer Selection

- Tokenization
  - Load `allenai/scibert` tokenizer:
    - python
    - Copy
    - Download

```
- from transformers import AutoTokenizer
    - tokenizer = AutoTokenizer.from_pretrained("allenai/scibert")
```

- Base Model
  - Start with `microsoft/phi-1_5` (small but strong for Q&A) or `mistral-7b` (if Kaggle GPU allows).
  - Pre-train with LoRA (low-rank adaptation) for efficiency:
    - python
    - Copy
    - Download

```
- from peft import LoraConfig
    - lora_config = LoraConfig(r=8, target_modules=["q_proj", "v_proj"])
```

4. Training & Validation (In-Notebook)

- Small-Scale Validation
  - Reserve 5% of papers for validation. Generate 100 Q&A pairs manually for testing.
- Metrics
  - Track:
    - Factual Accuracy: Use `rouge`/`bleu` vs. ground-truth answers.
    - Coherence: Heuristic checks (e.g., repetition count via regex).
- Early Stopping
  - Stop if loss plateaus for 3 epochs (use `EarlyStoppingCallback`).

5. Output Quality Assurance

- Post-Training Checks
  - Test with diverse prompts:
    - python
    - Copy
    - Download

- `inputs = tokenizer("What is a hardware wallet?", return_tensors="pt").to("cuda")`
- `outputs = model.generate(**inputs, max_new_tokens=100)`
    - `print(tokenizer.decode(outputs[0]))`
    - Critical Checks:
        - No hallucinations (answers must trace to papers).
        - No repetition/circular logic.

6. Export for Deployment

- Save the trained model to Kaggle Output or HuggingFace Hub:

```
python
Copy

Download
```

- `model.save_pretrained("my_pre-trained_model")`
- `tokenizer.save_pretrained("my_pre-trained_model")`
-