Below are some suggestions and considerations for both improving the training data and enhancing the code for a better student experience.

Improving and Expanding the Training Data

Right now, the training data is a collection of popular movie quotes, phrases, and cultural references. While it's fun and recognizable, it's also quite varied and somewhat random. A key reason language models produce more coherent text as they improve is because they learn patterns that are easier to identify in more consistent or thematically grouped data. If you add more data that:

1. **Forms Short Narratives or Stories:**
   Include some short, coherent paragraphs that tell a simple story—maybe 3–5 sentences each—about a character, a day at school, an adventure, etc. This will help the model produce more contextually coherent output when `window_size` is increased. For example:

   ```
   Alice woke up early and decided to explore the old forest.
   She walked past towering trees and listened to birds singing.
   By noon, she found a hidden stream and sat down to rest.
   As evening fell, she followed the fireflies back home.
   ```

   Adding multiple short stories like this provides patterns of context, continuity, and narrative flow.

2. **Include a Few Longer Sentences or Paragraphs with Repeated Themes:**
   The model currently sees mostly short, punchy lines. Include a handful of longer sentences or small paragraphs that repeat certain subjects or structures. For example:

   ```
   On a crisp autumn morning, the cat perched silently on the windowsill,
   watching leaves tumble in the wind and children laugh on their way to
   school.
   In the quiet library, pages turned softly, and a sense of calm understanding
   settled over the readers.
   ```

   Such data helps the model pick up on descriptive language and context clues.

3. **Add More Transitional and Contextual Phrases:**
   Right now, many phrases are stand-alone quotes. Add text that uses transitions like "However," "Later that day," "Meanwhile," "As a result," and so forth. This encourages the model to produce more logically connected sentences.

4. **Keep Some Familiar Quotes and Cultural References, But Group Them:**
   Maybe group the movie quotes by theme or genre so that when the model looks back a few words, it's more likely to find related thematic content. For example, cluster some fantasy references together (Lord of the Rings, Harry Potter, Game of Thrones), some science fiction references together (Star Wars, Star Trek, Marvel), and so on. This can help the model form mini "topic clusters" that produce more coherent responses as you tweak the window size and temperature.

By blending thematic clusters, short narratives, and a few longer descriptive passages, your dataset will give the model richer patterns to learn from. Students will see more coherent text emerge, especially by the time they reach the interactive phase.

Potential Improvements to the Code

1. **Adjusting `window_size` and Output Length Defaults:**
   For `01_basic_predictor.py`, no context is fine. For `02_markov_window.py`, consider picking a slightly higher default `window_size` (e.g., 2 instead of 1) and a reasonable `output_length` (like 20) to immediately show the benefit of increased context. Students can still experiment, but starting with slightly more context might give them a better initial “wow” moment.

2. **Highlighting the Effects of Changes More Clearly:**
   In `02_markov_window.py`, after generating text, you might print out a message comparing the result to what they saw in `01_basic_predictor.py`. This helps reinforce the concept that the improvement in coherence is due to the Markov approach.

3. **Introduce a Simple Tokenization Step (Optional):**
   While you’re mainly dealing with whitespace tokenization, you could use a simple library like `nltk` (no API key required) for slightly more robust tokenization if desired. This might make punctuation handling cleaner, resulting in more natural outputs.

   ```python
   # Example (in setup code, not necessarily required):
   import nltk
   nltk.download('punkt')
   from nltk.tokenize import word_tokenize
   ```

   Using `word_tokenize` on each line might yield cleaner word sequences for the Markov model. This is optional but can improve the learning experience slightly.

4. **Enhance the Interactive Phase (04_interactive.py):**

   - Allow students to input multiple starting words (if `window_size` > 1) or pick from a predefined set of starting phrases to guide the text generation toward something more coherent.
   - Print out some statistics: for example, after generation, show how many unique words the model considered at each step, or how the temperature affected word choice. This gives a more tangible sense of what’s changing under the hood.
   - If you’re open to additional Python libraries that don’t require API keys, consider using `collections.Counter` or `numpy` for slightly more sophisticated probability sampling. This wouldn’t drastically change the output quality but might give students more insights into how probabilities are computed and how changing temperature affects distribution sampling.

5. **Adding Comments and Instructions in the Code:** The code is already commented, but you might add more educational comments that remind students what’s happening at each step. For instance:

   ```python
   # Here we pick the next word based on the current 'window' of previous
   words.
   ```

```
    # If temperature is low, we bias towards the most frequent words.
    # If temperature is high, we add more randomness.
    # Students: Try changing these values and observe the results.
```

These in-code nudges help guide students to experiment more confidently.

Conclusion

- **Expanded and More Structured Training Data:** Helps the model learn better patterns, improving coherence.
- **Minor Code Tweaks and More Guidance:** Ensures students see clearer progress from random output to coherent text.
- **Optional Simple Libraries:** Like NLTK for tokenization—no API key required—could slightly improve token handling.

With these adjustments, by the time students reach `04_interactive.py`, they should see a noticeable improvement in the quality of generated text compared to the initial random output, making the exercise more rewarding and illustrating the concepts more clearly.