Q: Briefly describe the key limitation of TF-IDF that Word2Vec aimed to solve. What limitation of sequential models (like LSTMs) did the Transformer architecture address with its attention mechanism?

A: TF-IDF couldn't properly examine the semantic relationships between words; it treated "car" and "automobile" as completely unrelated terms although they generally refer to the same things. Word2Vec solved this by creating dense vector representations where these semantically similar words would appear close together in vector space. Sequential models (LSTMs) struggled with long-range dependencies and couldn't properly be parallelized. Transformer architecture's' attention mechanism solved this by allowing direct connections between any sequenced words without paying mind to distance, enabling parallel processing and better capturing relationships between words.

Q: What was the significance of BERT's "bidirectional" pre-training compared to earlier models?

A: Earlier models like GPT processed text only from left-to-right, which left them with a very limited context without being able to use retrospect. BERT's bidirectional approach allowed it to interpret a word after it has processed the words ahead and behind it to understand the context and meaning. By masking random tokens and asking the model to predict them using all surrounding contexts, BERT developed a much deeper understanding of language that captured meaning from both directions.

Q: Create a table summarizing the key differences (pros/cons) between using a locally run Sentence-BERT model versus the OpenAI text-embedding-ada-002 API.

A:

Feature	Sentence-BERT	OpenAI Ada-002
Hosting	Local hardware	Cloud API
Cost	Free + compute costs	Pay-per-token
Privacy	High (data stays local)	Lower (data sent to OpenAI)
Setup	Complex	Simple API calls
Max Input	~512 tokens	8191 tokens

Customization	Can be fine-tuned	Fixed model
Performance	Hardware-dependent	Fast, consistent

Q: Imagine you have a large PDF textbook (1000 pages) that you want to use for RAG. Why is chunking necessary? Describe two different chunking strategies you could use. What factors would influence your choice of chunk size and overlap?

A: Chunking is necessary because embedding models have token limits (512-8191) that can't handle a 1000 page textbook. It also improves retrieval precision by returning only the relevant sections.

Two strategies:

1. Structure-based chunking: Split by chapters, sections, and subsections to preserve the textbook's organization.
2. Recursive text splitting: Use a hierarchy of separators (chapter breaks, paragraphs, sentences) with overlap between chunks.

Factors influencing chunk size/overlap:

- Model token limits
- Content density (technical vs. narrative)
- Query complexity
- Cross-reference requirements
- Retrieval precision vs. context needs
- Available computational resources

Q: For each scenario, which embedding approach (OpenAI Ada vs. Local SBERT) might be more suitable and why?

A:

- **Startup chatbot for website FAQs:** OpenAI Ada - Faster implementation, no setup overhead, incremental payments work for prototypes, and FAQ data isn't too sensitive.
- **Hospital system for patient records:** Local SBERT - Privacy is critical for medical data, keeps sensitive information in-house for regulatory compliance, and can be specialized for medical use.

- **Solo developer's note-taking app:** Local SBERT - One-time setup vs concurrent API costs, keeps notes private and accessible offline and can run effectively on a powerful desktop.