Predicting the Next Word

From Shannon to ChatGPT

Test Compilation - Chapter 3

Contents

Chapter 3

Tokenization

In this chapter, we advance next-word prediction by:

- Understanding how tokenization defines the vocabulary $\mathcal{V}$ over which language models predict
- Exploring subword algorithms (BPE, WordPiece, SentencePiece) that balance vocabulary coverage and computational efficiency
- Analyzing the trade-offs between vocabulary size, fertility, and downstream model performance
- Learning to handle rare words, novel entities, and multilingual text through byte-level fallback mechanisms

3.1 Tokenization and Next-Word Prediction

Before a language model can predict the next word, it must answer a more fundamental question: what exactly constitutes a "word" in the first place? In Chapter **??**, we introduced the vocabulary $\mathcal{V}$ as the set of all possible prediction targets, computing probabilities $P(w_t \mid w_1, \ldots, w_{t-1})$ over this discrete set. However, we deferred a critical question: how do we construct this vocabulary from raw text? The answer lies in tokenization, the process of segmenting continuous text into discrete units called tokens. This seemingly simple preprocessing step has profound implications for everything that follows. The choice of tokenization algorithm determines the size of $\mathcal{V}$, affects how the model handles rare or novel words, influences the computational cost of training and inference, and ultimately shapes what patterns the model can learn from data. Every modern language model, from GPT to BERT to LLaMA, begins with tokenization, making it one of the most consequential design decisions in natural language processing. Understanding tokenization is essential for anyone seeking to build, analyze, or improve language models.

Consider the challenge of predicting the next word after the context "The researcher studied electroencephalography and". A word-level tokenizer might never have seen "electroencephalography" in training data, forcing the model to output an unknown token symbol and effectively giving up on understanding this context. A character-level tokenizer avoids this problem by treating each letter as a separate token, but now the model must process 22 tokens just for this single word, dramatically increasing computational cost and making it harder for attention mechanisms to capture long-range dependencies across hundreds of positions. Subword tokenization offers a middle ground: the word might be split into meaningful pieces like "electro", "encephalog", and "raphy", each appearing frequently enough in training to have learned representations, while keeping the sequence length manageable. The probability computation $P(w_t \mid \mathbf{c})$ now operates over these subword units, fundamentally changing what the model predicts. This chapter explores the spectrum of tokenization approaches, from simple word splitting to sophisticated learned subword vocabularies, examining the trade-offs that have led modern language models to converge on subword tokenization as the dominant paradigm.

Figure 3.1: Tokenization converts raw text into discrete token IDs that index into the vocabulary $\mathscr{V}$. The choice of tokenization algorithm—word-level, character-level, or subword—determines what units the language model learns to predict. Each approach represents a different trade-off between vocabulary size, sequence length, and the ability to handle unseen words.

3.2 Word-Level Tokenization and Its Limitations

The most intuitive approach to tokenization treats whitespace and punctuation as natural boundaries between words, mapping directly to how humans perceive written language as composed of discrete word units. Given the sentence "The cat sat on the mat.", a simple word-level tokenizer splits on spaces and separates punctuation to produce the token sequence ["The", "cat", "sat", "on", "the", "mat", "."]. This approach aligns with human intuition about language structure and was the dominant paradigm in early natural language processing systems, from rule-based parsers of the 1970s through the statistical models of the 1990s and early 2000s. The vocabulary $\mathcal{V}$ consists of all unique words observed in the training corpus, and the language model learns to predict $P(w_t \mid w_1, \ldots, w_{t-1})$ where each w_i is a complete word. For languages with clear word boundaries like English, this approach seems natural and produces reasonable vocabulary sizes when trained on large corpora, typically ranging from 50,000 to 200,000 unique word types depending on the corpus size and domain coverage.

However, word-level tokenization suffers from a fundamental problem known as the out-of-vocabulary (OOV) dilemma. Natural language exhibits a long-tailed distribution of word frequencies: a small number of words appear extremely often, while the vast majority appear rarely. Zipf's law, which we encountered in Chapter **??**, tells us that the frequency of a word is inversely proportional to its rank. In a typical English corpus, approximately half of all unique word types appear only once, and many more appear fewer than five times. When the model encounters a word it has never seen during training, it must map that word to a special unknown token [UNK], losing all information about the original word. This problem becomes severe when processing text from domains not represented in training data, such as technical documents, social media with creative spellings, or text containing proper nouns like brand names or personal names that emerged after the training data was collected.

The vocabulary explosion problem becomes even more severe in morphologically rich languages, where the combinatorial possibilities of word formation vastly exceed those of analytic languages like English. Consider Finnish, Turkish, or Hungarian, where words are formed by combining stems with numerous prefixes and suffixes to express grammatical relationships, generating an exponentially larger space of valid word forms. A single Finnish verb can have thousands of inflected forms: "juoksentelisinkohan" means "I wonder if I should run around aimlessly." In agglutinative languages like Turkish, the word "evlerinizdeki" (meaning "that which is in your houses") combines multiple morphemes into a single orthographic word that would be a phrase in English. German compounds nouns freely: "Rindfleischetikettierungsüberwachungsaufgabenübertragungsgesetz" is a single word describing a law about beef labeling supervision. For word-level tokenization, each unique combination must be stored as a separate vocabulary entry, with no parameter sharing between related forms. A vocabulary that covers 99% of English words might cover less than 90% of Finnish or Turkish text, and achieving similar coverage would require vocabularies orders of magnitude larger, making the embedding matrix computationally prohibitive and storage requirements impractical. This fundamental limitation motivated the development of subword tokenization methods that can decompose complex words into reusable pieces, sharing representations across morphologically related forms and enabling effective handling of productive word formation processes.

Figure 3.2: Word-level tokenization faces three interconnected problems: (1) the long-tailed frequency distribution means most words are rare, (2) out-of-vocabulary words must be mapped to [UNK] tokens losing all information, and (3) morphologically rich languages generate exponentially more word forms than analytic languages like English.

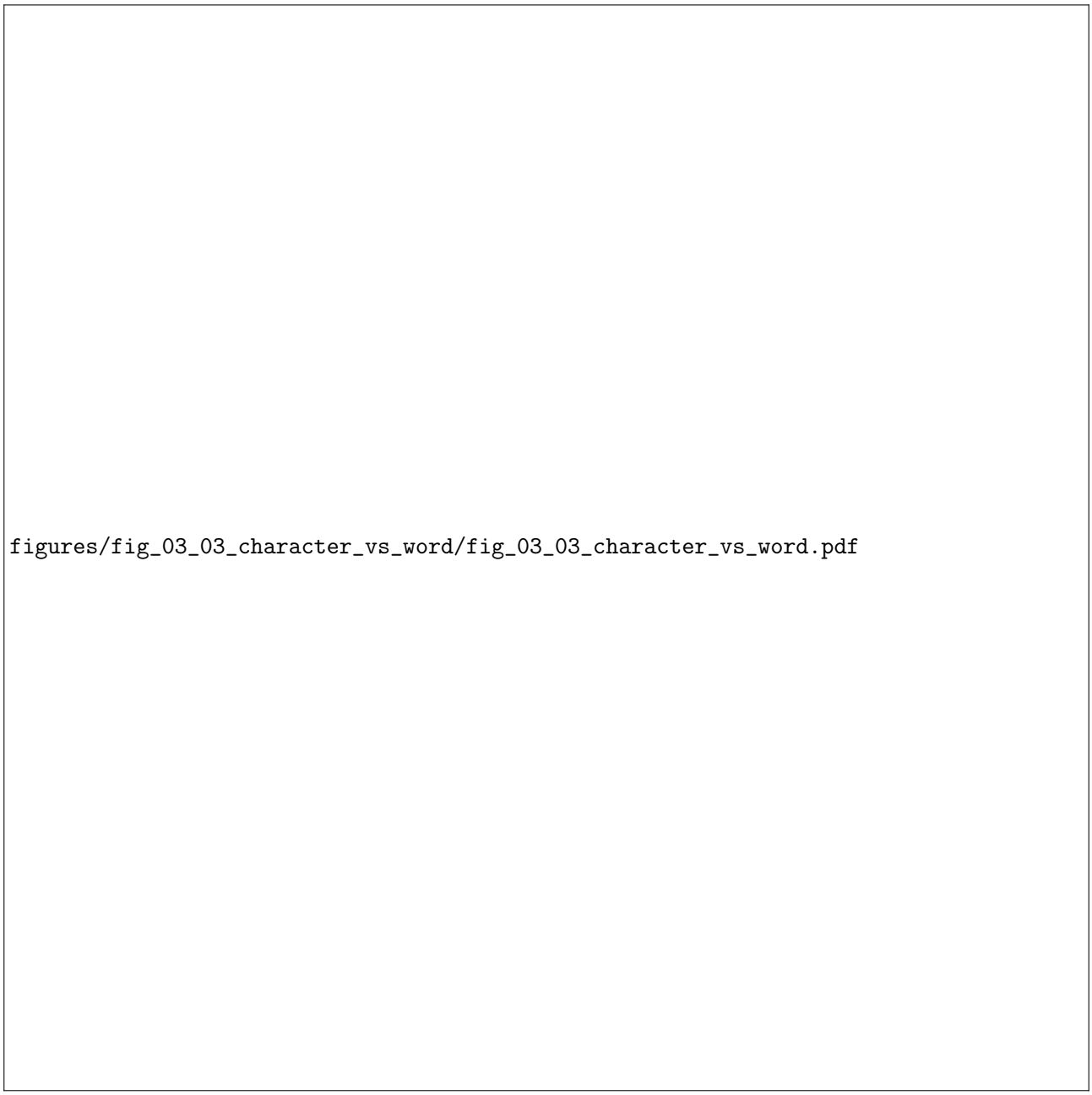

Figure 3.3: The same sentence tokenized at character level versus word level illustrates the fundamental trade-off: character-level tokenization eliminates out-of-vocabulary words entirely but produces much longer sequences that increase computational cost and make it harder for models to capture long-range dependencies.

3.3 Character-Level and Byte-Level Approaches

One radical solution to the out-of-vocabulary problem is to abandon words entirely and operate at the level of individual characters. A character-level tokenizer for English needs only about 100 tokens to represent the full ASCII character set, or perhaps 256 tokens to cover the extended ASCII range including accented characters and special symbols. The vocabulary $\mathcal{V}$ becomes tiny and fixed: we never encounter an unknown character because all possible characters are enumerated in advance. The word "electroencephalography" that caused problems for word-level tokenization is simply represented as the sequence ["e", "l", "e", "c", "t", "r", "o", ...], each character having a well-defined entry in the vocabulary. Novel words, creative spellings, technical terms, and proper nouns all decompose naturally into their constituent characters without requiring any special handling. This elegant solution to the OOV problem was explored in early neural language models and continues to find applications in specific domains where character-level patterns matter, such as morphological analysis, spelling correction, and code generation where individual characters carry syntactic significance.

The cost of character-level tokenization is dramatically increased sequence length. A typical English word contains about 5 characters on average, so a character-level model must process roughly five times as many tokens as a word-level model for the same text. This matters enormously for computational efficiency, particularly with attention-based architectures where the computational cost scales quadratically with sequence length, meaning that a 5x increase in sequence length produces a 25x increase in attention computation. A sentence that word-level tokenization represents with 20 tokens might require 100 tokens at the character level. Training on documents of reasonable length becomes prohibitively expensive, and the model must learn to compose meaning across much longer distances. While a word-level model can directly associate "cat" with feline concepts, a character-level model must learn that the sequence ["c", "a", "t"] forms a meaningful unit distinct from ["c", "a", "r"] or ["c", "a", "p"], and that this unit carries semantic content about small furry animals. This hierarchical composition must be learned entirely from data, without any explicit supervision about word boundaries.

Byte-level tokenization pushes this idea to its logical extreme by treating text as a sequence of raw bytes. Every Unicode character, regardless of script or language, has a well-defined UTF-8 encoding as a sequence of 1 to 4 bytes. The vocabulary contains exactly 256 possible byte values, providing complete coverage of any text that can be represented digitally. This includes not only all human writing systems—Latin, Cyrillic, Arabic, Chinese, Japanese, Korean, and hundreds more—but also emoji, mathematical symbols, and any other Unicode characters that might appear in training or inference data. The byte-level approach guarantees that no input can ever produce an unknown token, making it truly universal and future-proof against new characters added to Unicode. GPT-2 pioneered a practical implementation using byte-level BPE, treating bytes as the base units from which larger subword tokens are built, combining the universality of byte-level representation with the efficiency gains of learning common byte sequences. This hybrid approach has become the standard for modern large language models, providing robustness without sacrificing computational efficiency.

Figure 3.4: Tokenization exists on a spectrum from characters (small vocabulary, long sequences) to words (large vocabulary, short sequences). Subword tokenization occupies the middle ground, learning to segment text into units that balance vocabulary size against sequence length while maintaining the ability to represent any input through composition.

3.4 Subword Tokenization

Subword tokenization emerged as the dominant paradigm in modern language modeling by finding a principled middle ground between the extremes of word-level and character-level approaches. The core insight is that while words vary enormously in frequency, certain sub-word units—prefixes, suffixes, and stems—recur across many words and can serve as efficient building blocks. The word "unhappiness" can be decomposed into ["un", "happi", "ness"], where each piece appears in many other words: "un" in "undo", "unclear", "unfair"; "ness" in "darkness", "kindness", "awareness". By learning a vocabulary of such reusable pieces, subword tokenization achieves reasonable sequence lengths while maintaining the ability to represent any word through composition. Several algorithms have been developed to learn these vocabularies automatically from data, each with slightly different properties and trade-offs. The key algorithms—Byte Pair Encoding, WordPiece, SentencePiece, and Unigram Language Model tokenization—share the fundamental goal of discovering useful subword units but differ in how they construct and select vocabulary items.

3.4.1 Byte Pair Encoding (BPE)

Byte Pair Encoding, originally developed as a data compression algorithm in 1994, was adapted for neural machine translation by Sennrich, Haddow, and Birch in 2016 [Sennrich et al., 2016]. The algorithm begins with a vocabulary containing only individual characters (or bytes, in byte-level BPE), then iteratively merges the most frequent adjacent pair of tokens to create a new vocabulary entry. Starting with the word "lower" represented as ["l", "o", "w", "e", "r"], if the pair ("e", "r") appears most frequently across the training corpus, it is merged to create a new token "er", and all occurrences of the sequence are replaced. The vocabulary now contains ["l", "o", "w", "e", "r", "er"], and "lower" is represented as ["l", "o", "w", "er"]. This process repeats for a predetermined number of merge operations—typically 30,000 to 50,000—gradually building up common subwords, whole words, and even multi-word sequences. The order of merges is recorded and must be applied in the same sequence during inference to ensure consistent tokenization between training and deployment.

The beauty of BPE lies in its simplicity and its connection to data compression. Each merge operation reduces the total number of tokens needed to represent the training corpus by replacing two tokens with one wherever the merged pair occurs. Frequent words quickly become single tokens: after enough merges, common words like "the", "and", "is" will be represented as single vocabulary entries because their constituent character sequences appear together so often. Rare words, conversely, remain decomposed into smaller pieces that they share with other words. The word "electroencephalography" might be tokenized as ["electro", "encephal", "ography"], each piece appearing in other medical or scientific terms. This automatic discovery of morphological structure emerges purely from frequency statistics, without any linguistic knowledge being provided to the algorithm. The algorithm makes no distinction between morphological boundaries and accidental letter combinations; both emerge from the same frequency-based merging process, yet the result often aligns surprisingly well with meaningful linguistic units.

3.4.2 WordPiece

WordPiece, developed at Google for their neural machine translation system [Wu et al., 2016], takes a similar iterative approach to BPE but uses a different criterion for selecting which pairs to merge, grounding the decision in statistical principles rather than simple frequency counting. Instead of simply choosing the most frequent pair, WordPiece selects the pair that maximizes the likelihood of the training data when the merge is applied. Formally, if we consider merging symbols x and y to create xy, WordPiece computes the increase in log-likelihood: $\log P(xy) - \log P(x) - \log P(y)$. This criterion favors merging pairs where the combined token is much more likely than would be predicted by the independent frequencies of its parts, capturing genuine co-occurrence patterns rather than just raw frequency. A pair like ("t", "h") might be very frequent in English, but ("t", "he") captures a stronger dependency because "the" is an extremely common word. This mutual information-like criterion provides a more principled approach to deciding which merges are most valuable.

The practical difference between BPE and WordPiece is subtle but meaningful in terms of the vocabularies they produce. BPE's frequency-based merging tends to create tokens that reflect simple surface statistics,

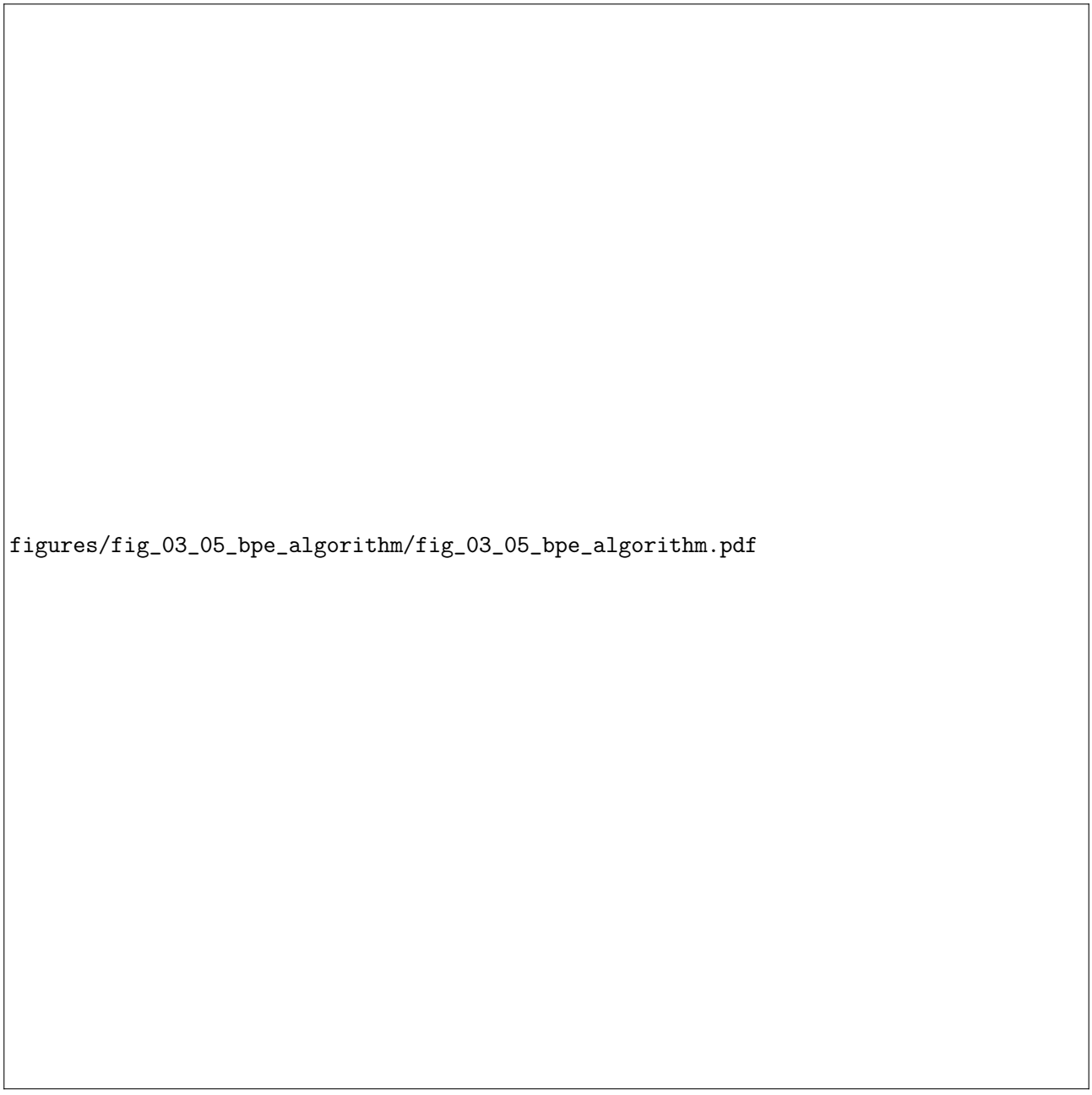

Figure 3.5: Byte Pair Encoding iteratively merges the most frequent adjacent token pair. Starting from a character vocabulary, each merge operation adds one new token and reduces the total sequence length across the corpus. The algorithm terminates after a fixed number of merges, which determines the final vocabulary size.

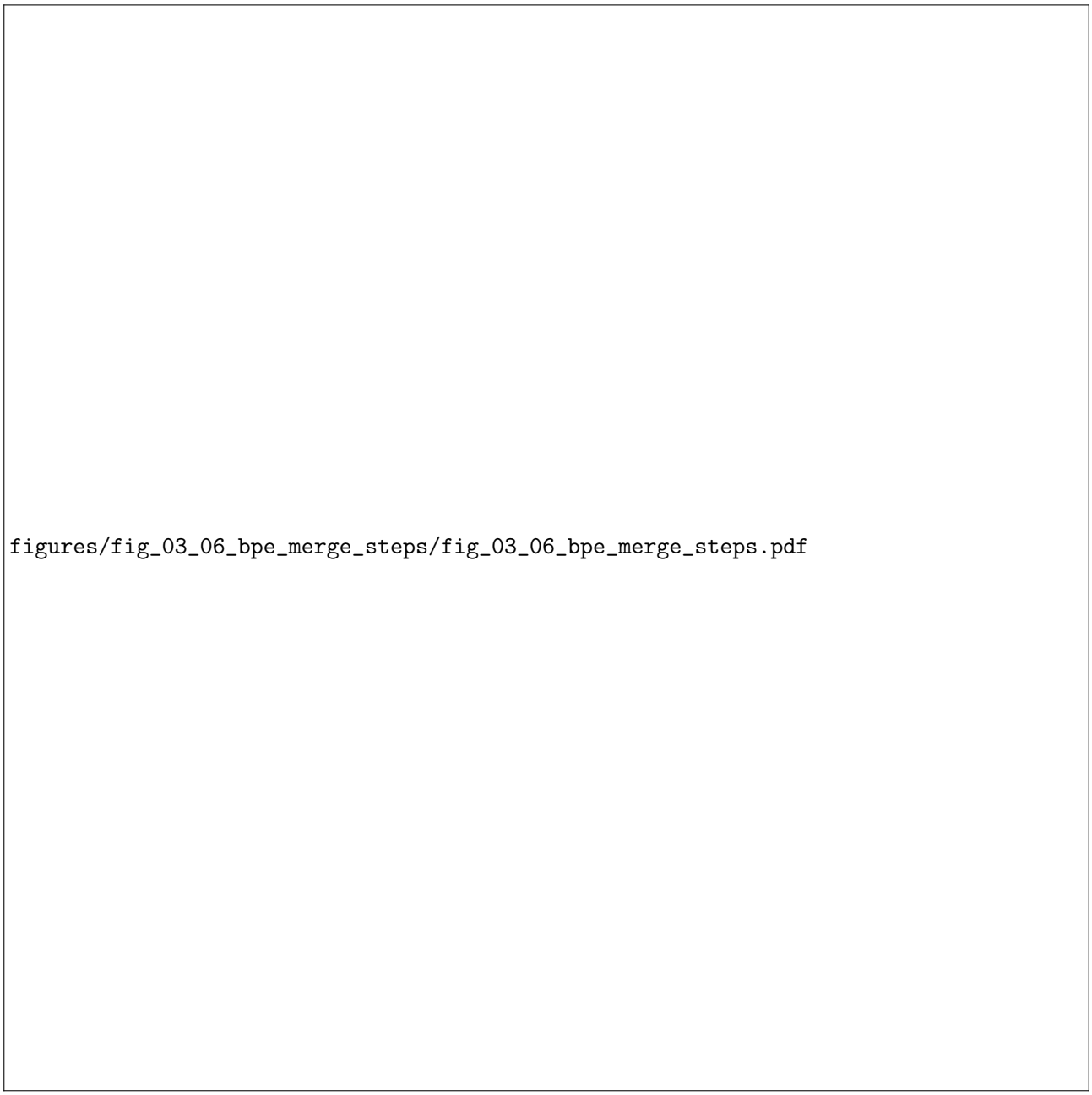

Figure 3.6: A detailed trace of BPE merges on a sample corpus shows how the algorithm progressively builds larger tokens from frequent character pairs. Each step displays the current vocabulary, pair frequencies, and the resulting tokenization of the corpus.

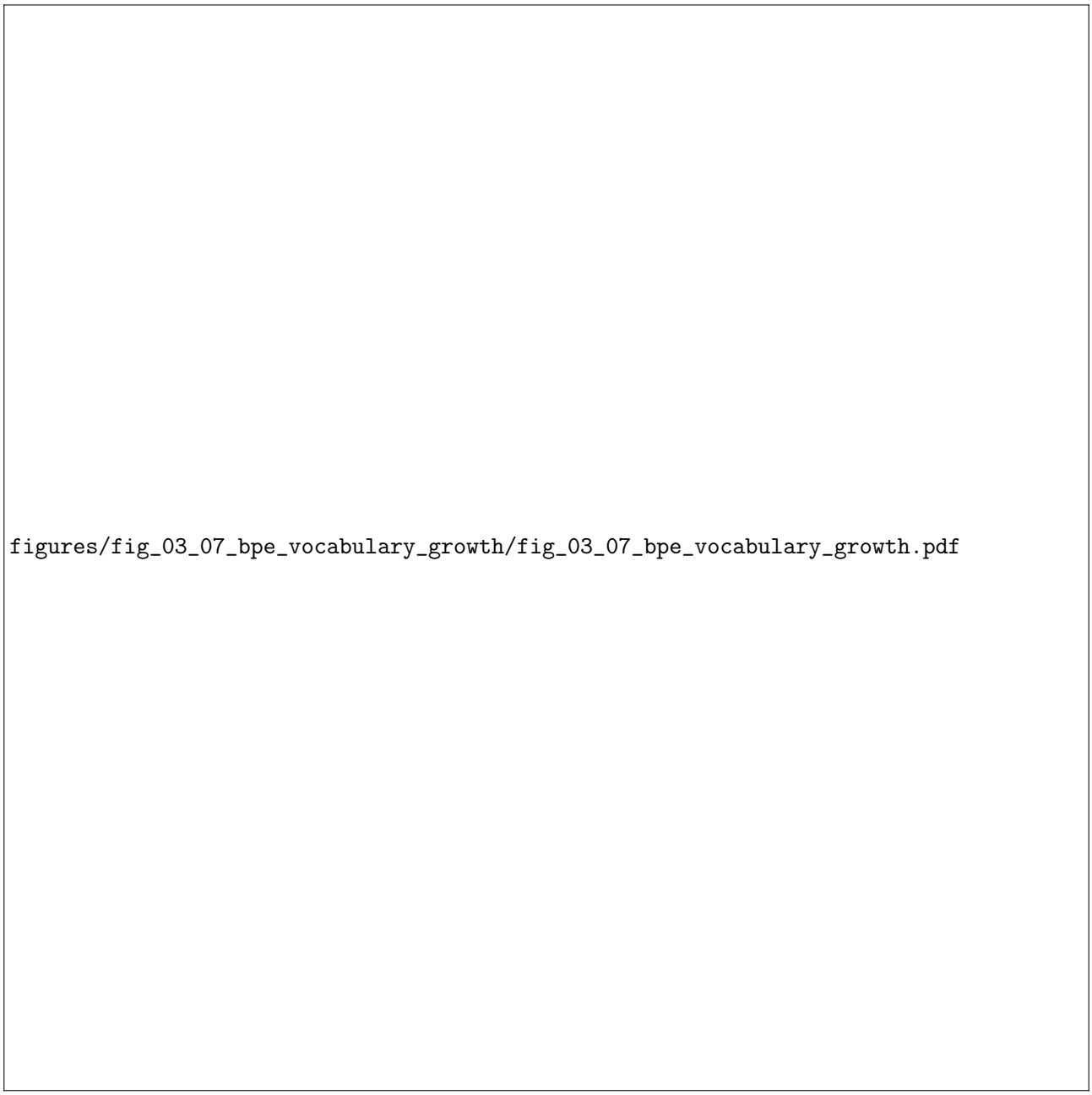

Figure 3.7: As BPE performs more merge operations, the vocabulary grows linearly while the total number of tokens needed to represent the corpus decreases. This trade-off between vocabulary size and sequence length is the fundamental parameter that practitioners must tune.

while WordPiece's likelihood-based criterion can discover more linguistically meaningful units by focusing on tokens that co-occur more than chance would predict. WordPiece became widely known through its use in BERT and subsequent transformer models, where it typically produces vocabularies of around 30,000 tokens. One distinctive feature of WordPiece is its use of the "##" prefix to indicate subword tokens that continue a word: "playing" might be tokenized as ["play", "##ing"], making it explicit that "##ing" is not a word-initial token. This notation helps with detokenization and makes the tokenization more interpretable for humans examining model behavior, while also providing the model with information about whether a subword begins a new word or continues an existing one, potentially improving the model's ability to handle word boundaries during prediction.

Figure 3.8: WordPiece selects merges based on likelihood improvement rather than raw frequency. The algorithm also marks continuation tokens with a special prefix (##) to distinguish word-initial from word-internal subwords.

3.4.3 SentencePiece

SentencePiece, developed by Kudo and Richardson at Google [Kudo and Richardson, 2018], addresses a fundamental limitation shared by both BPE and WordPiece: their reliance on language-specific pre-tokenization. Standard implementations of these algorithms assume that the input has already been split into words by whitespace, then learn subword vocabularies within those word boundaries. This assumption works reasonably well for languages like English that use spaces between words, but fails completely for languages like Chinese, Japanese, and Thai where words are not separated by whitespace and must be identified through other means such as dictionary lookup or statistical segmentation, which themselves require language-specific resources. Even for English, pre-tokenization introduces language-specific rules for handling punctuation, contractions, and other edge cases that may not generalize across languages or domains, creating a brittle dependency on language-specific preprocessing that complicates truly multilingual systems and can introduce subtle bugs when processing text that violates expected conventions or contains mixed-language content where multiple scripts appear together.

SentencePiece treats the input as a raw stream of Unicode characters, with whitespace treated as just another character rather than a word boundary, eliminating the need for language-specific preprocessing entirely. The underscore character (displayed as a special symbol) replaces spaces in the input, so "Hello world" becomes "_Hello_world". Subword learning then proceeds without any assumption about word boundaries, discovering segments that may cross what humans would consider word boundaries. This approach makes SentencePiece truly language-agnostic: the same algorithm can be applied to English, Japanese, Thai, or any other language without modification, using identical code paths for all languages. The framework supports both BPE and unigram language model algorithms for the actual vocabulary learning, providing flexibility in choosing the underlying tokenization method while maintaining the language-independent preprocessing. This flexibility has made SentencePiece the tokenization framework of choice for many multilingual and cross-lingual language models including mBERT, XLM-R, and T5.

3.4.4 Unigram Language Model Tokenization

The unigram language model approach to tokenization, also developed by Kudo and available in SentencePiece, takes a fundamentally different approach from the bottom-up merging of BPE and WordPiece. Instead of starting small and building up, the unigram method starts with a large vocabulary of candidate subwords and iteratively removes the least useful ones, proceeding top-down rather than bottom-up. The algorithm begins by generating a large set of potential subword candidates—often all substrings up to a certain length that appear in the corpus, potentially numbering in the millions—then fits a unigram language model where each subword has an independent probability. Given this model, it computes the tokenization of the training corpus that maximizes likelihood, then identifies which vocabulary items contribute least to this likelihood. These low-impact items are pruned, the model is re-estimated on the reduced vocabulary, and the process repeats until reaching the target vocabulary size, typically requiring dozens of pruning iterations.

The key insight of unigram tokenization is that the same word can have multiple valid tokenizations, and the model explicitly represents this ambiguity rather than committing to a single deterministic segmentation as BPE and WordPiece do. The word "internationalization" might be tokenized as ["international", "ization"] or ["inter", "national", "ization"] or many other valid segmentations, each with a probability under the unigram model that reflects the learned subword frequencies from the training corpus. During training, this ambiguity can be exploited through subword regularization: rather than always using the single most likely tokenization, the model samples from the distribution of possible tokenizations, exposing it to different segmentations of the same word across different training examples. This regularization acts as a powerful form of data augmentation, making the model more robust to variations in how words are tokenized and reducing overfitting to specific subword boundaries that might be artifacts of the tokenization algorithm rather than meaningful linguistic structure.

Figure 3.9: SentencePiece treats whitespace as a regular character (represented by underscore) and learns subword vocabulary directly from raw text without pre-tokenization. This makes it applicable to any language regardless of word boundary conventions.

Figure 3.10: The unigram approach assigns probabilities to subwords and can represent multiple valid tokenizations of the same input. This probabilistic view enables subword regularization, where training samples different tokenizations to improve robustness.

3.5 Vocabulary Size and Design Decisions

Choosing the vocabulary size $|\mathcal{V}|$ is one of the most important practical decisions when training a tokenizer and the language model that depends on it. Common vocabulary sizes range from 8,000 tokens for smaller models to 50,000 or more for large multilingual systems, with 32,000 being a popular choice for many English-focused models such as GPT-2 and its successors. This choice affects multiple aspects of the system: the size of the embedding matrix (which scales linearly with vocabulary size and can represent a significant fraction of total model parameters for smaller models), the average sequence length (which decreases as vocabulary grows), and the model's ability to represent rare or domain-specific terms without resorting to fine-grained character-level decomposition. There is no universally optimal vocabulary size; the best choice depends on the languages covered, the domains of interest, the available computational budget, and the trade-off between model capacity and inference speed that practitioners must navigate based on deployment requirements.

The concept of fertility provides a useful lens for understanding vocabulary size trade-offs. Fertility is defined as the average number of tokens produced per word when text is tokenized, measuring how efficiently the tokenizer compresses text. A word-level tokenizer has fertility of exactly 1.0 by definition, while a character-level tokenizer for English has fertility of approximately 5.0 (the average word length). Subword tokenizers fall somewhere in between: a BPE tokenizer with 32,000 vocabulary might achieve fertility of 1.2 to 1.5 on English text, meaning most words are single tokens but some are split into two or three pieces. Fertility varies dramatically across languages: the same tokenizer might have fertility of 1.3 on English, 1.8 on German (with its compound nouns), and 2.5 or higher on morphologically rich languages like Finnish. Lower fertility means shorter sequences and faster processing, but requires larger vocabularies and embedding matrices; this trade-off must be carefully balanced when designing tokenizers for specific applications.

Beyond the learned vocabulary, modern language models reserve special tokens for structural purposes that go beyond simple text representation. The [UNK] (unknown) token handles the rare case where byte-level fallback is not available and an input cannot be tokenized, serving as a last-resort placeholder. The [PAD] (padding) token fills sequences to uniform length when batching inputs of different sizes, allowing efficient parallel processing on GPUs by ensuring all sequences in a batch have identical dimensions. For models trained with masked language modeling like BERT, the [MASK] token replaces words that the model must predict during training, creating the training signal for bidirectional context learning. Sequence boundary tokens like [CLS] (classification) and [SEP] (separator) provide explicit markers for the start of sequences and boundaries between sentence pairs, with [CLS] often serving as an aggregate representation of the entire sequence for classification tasks. The [BOS] (beginning of sequence) and [EOS] (end of sequence) tokens serve similar purposes in autoregressive models like GPT, marking where generation should start and when it should stop, essential for controlling generation during inference.

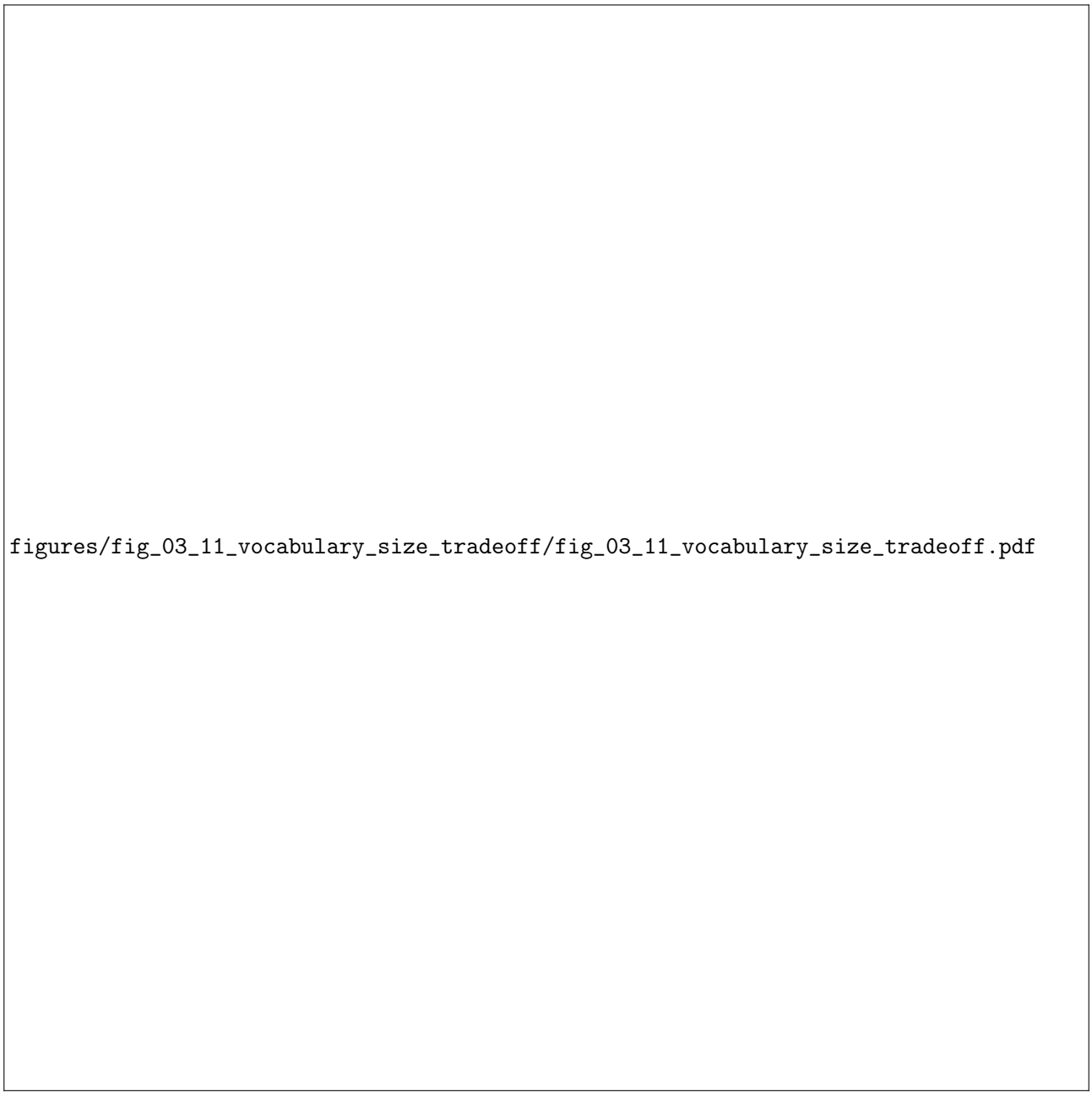

Figure 3.11: Larger vocabularies require more parameters for the embedding matrix but produce shorter token sequences. The optimal vocabulary size balances these competing concerns based on model architecture and computational constraints.

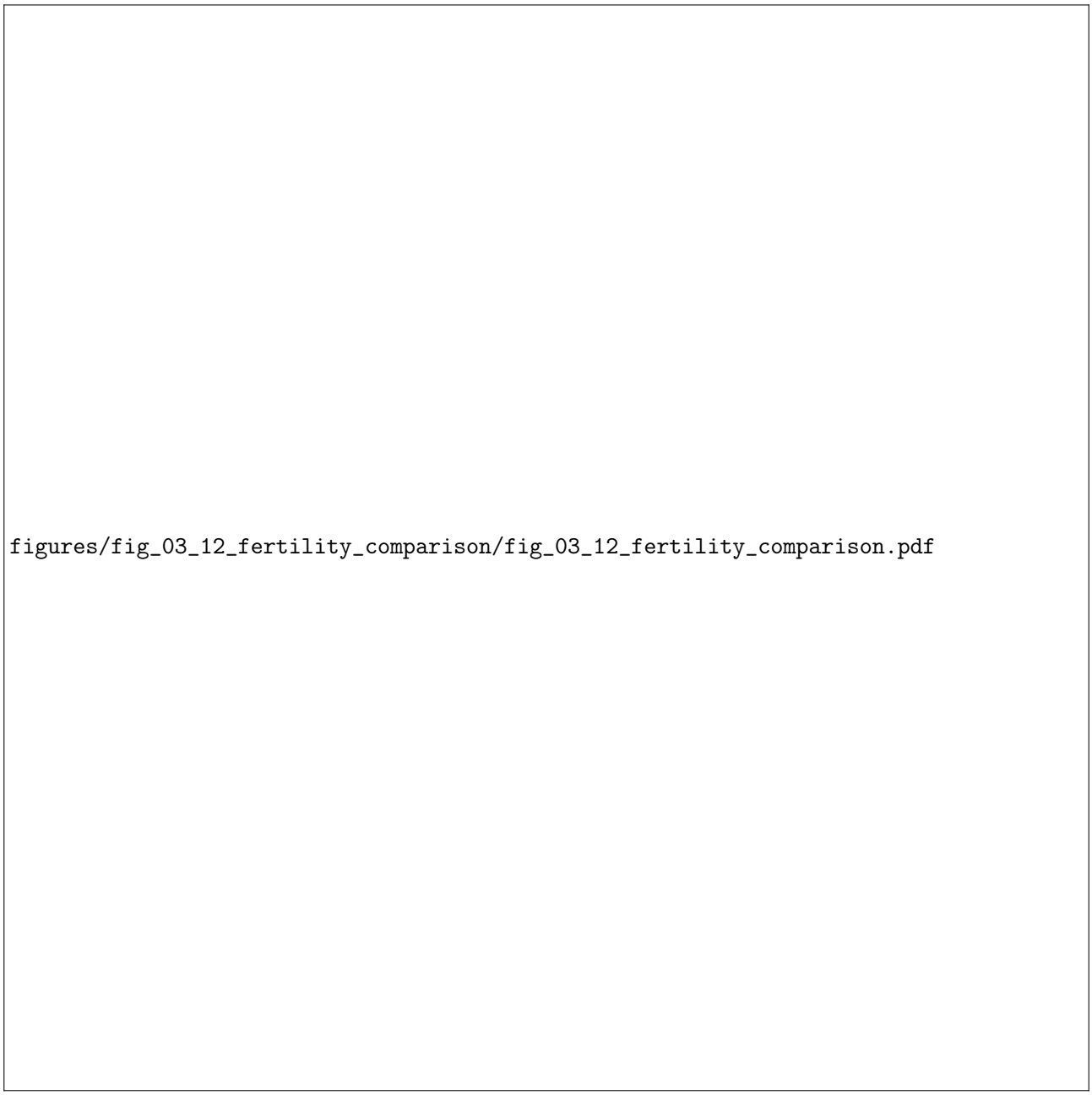

Figure 3.12: Fertility varies significantly across languages with the same tokenizer. English typically achieves the lowest fertility, while morphologically rich languages like Finnish or agglutinative languages like Turkish require more tokens per word.

Figure 3.13: Modern tokenizers use byte-level fallback to guarantee complete coverage: any character that cannot be tokenized as a learned subword is represented as a sequence of byte tokens, ensuring no input ever maps to [UNK].

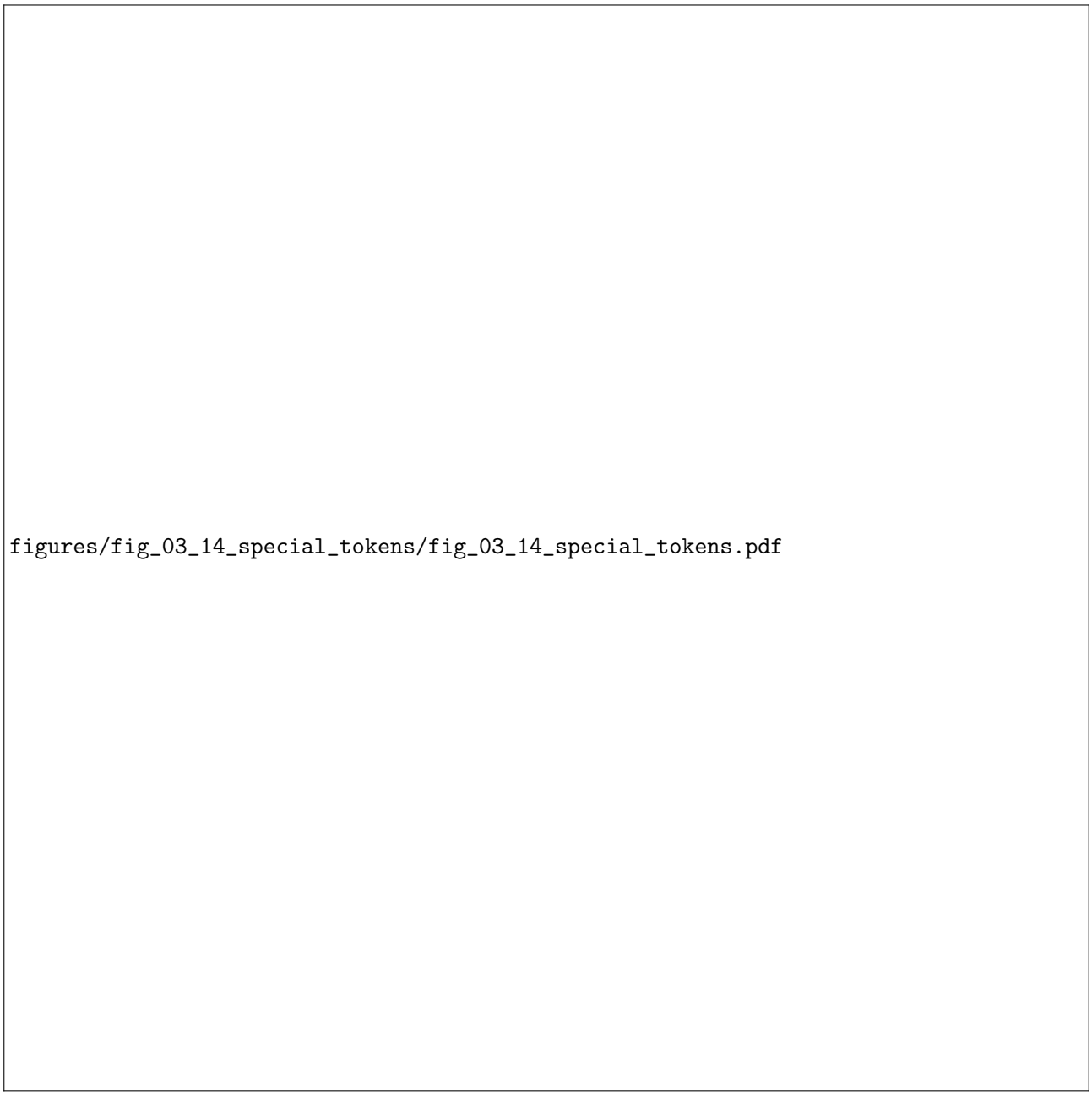

Figure 3.14: Special tokens provide structural information beyond text content. Each serves a specific purpose in model training or inference, from marking sequence boundaries to enabling masked language modeling.

3.6 Evaluating Tokenization Quality

Evaluating tokenization quality requires metrics that capture different aspects of how well a tokenizer serves the downstream language modeling task. The most direct metrics are fertility and compression ratio, which measure how efficiently the tokenizer represents text from complementary perspectives. Fertility, as discussed above, counts tokens per word, while compression ratio compares the number of bytes in the original text to the number of tokens produced. A tokenizer with compression ratio of 4 means that, on average, each token represents 4 bytes of the original text, effectively compressing the input by a factor of 4 in terms of sequence length. Higher compression ratios indicate more efficient tokenization, but this must be balanced against vocabulary size: trivially, a vocabulary containing every possible word would achieve compression ratio equal to average word length, but would require an impossibly large embedding matrix and would fail on any word not seen during vocabulary construction.

Coverage metrics assess how well the tokenizer handles the full range of inputs it might encounter, measuring the tokenizer's ability to efficiently represent diverse text from different languages and domains without excessive fragmentation into small pieces. For tokenizers without byte-level fallback, coverage measures what percentage of tokens in test data can be represented without mapping to [UNK], a critical metric that directly impacts model performance on out-of-distribution text. Even with perfect coverage guaranteed by byte fallback, coverage-like metrics remain useful: a tokenizer that represents most Chinese characters as multi-byte sequences rather than single tokens will have poor effective coverage for Chinese text, even though it technically produces valid output without any unknown tokens. Measuring the percentage of characters or words that tokenize to single tokens, rather than requiring multiple tokens, provides insight into how well the tokenizer serves different languages or domains. A tokenizer trained primarily on English text might achieve 95% single-token coverage on English words but only 30% on Japanese text, revealing significant bias toward the training language distribution that may substantially harm multilingual model performance and create unfair computational costs for non-English users.

The ultimate evaluation of tokenization quality comes from downstream task performance, which captures effects that simpler metrics cannot adequately measure. Two tokenizers with similar compression ratios and coverage might produce very different results when used to train language models on the same data. Factors that are difficult to capture in simple metrics—such as whether token boundaries align with morpheme boundaries, whether semantically related words share subword tokens, or whether the tokenization is stable across minor variations in input—can significantly affect model quality in ways that only become apparent through end-to-end evaluation on real tasks. Practitioners typically evaluate tokenizers by training small proxy models and measuring perplexity or downstream task accuracy, using these results to guide decisions about vocabulary size and tokenization algorithm for the final large-scale training run. This empirical approach to tokenizer selection reflects the difficulty of predicting a priori which tokenization choices will work best for a given application or language distribution.

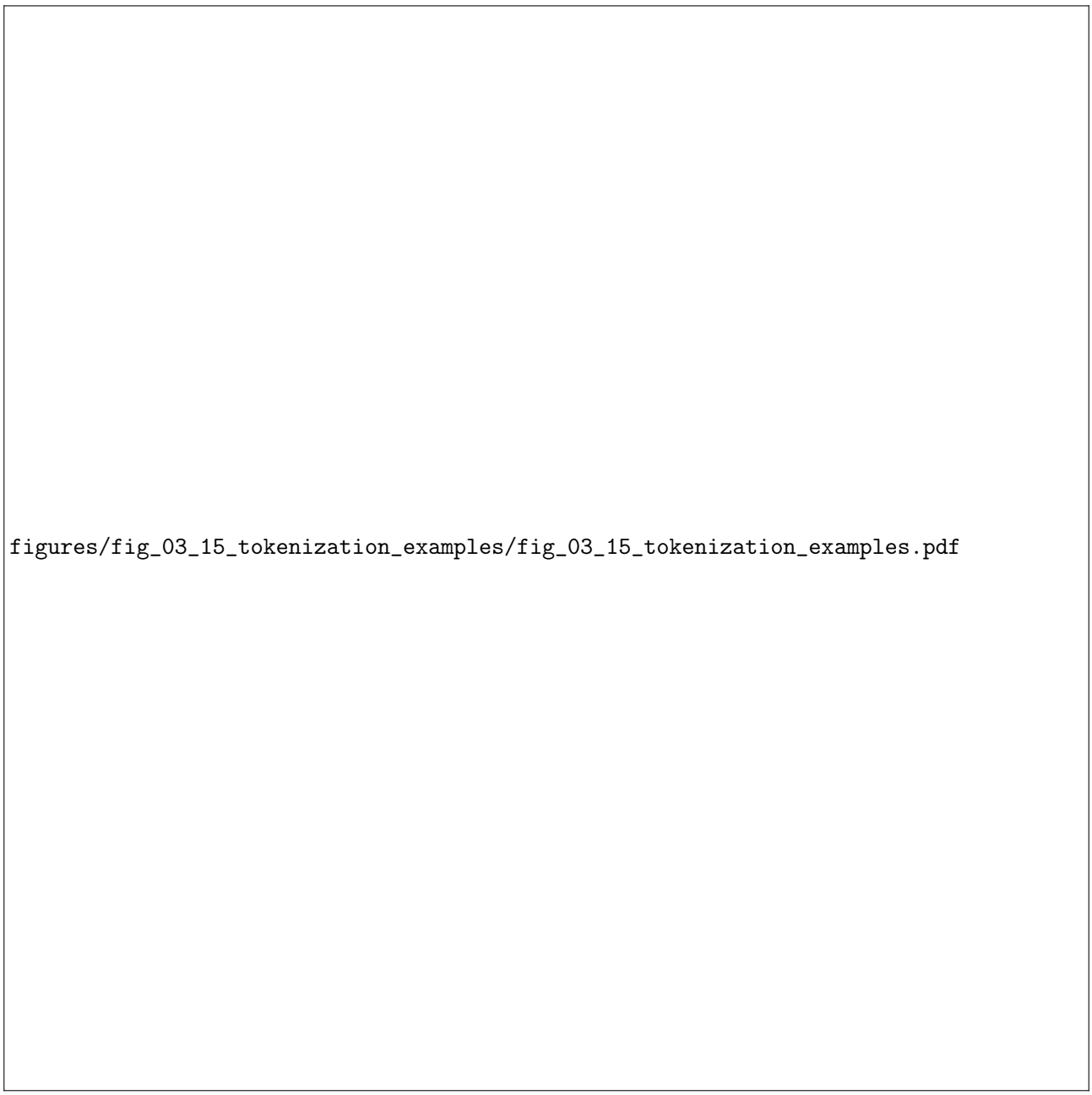

Figure 3.15: The same text tokenized by word-level, character-level, BPE, and WordPiece tokenizers illustrates the different trade-offs each method makes between vocabulary size, sequence length, and linguistic meaningfulness of token boundaries.

Figure 3.16: A tokenizer trained on predominantly English data shows dramatically different performance across languages. Languages with different scripts, morphological complexity, or word boundary conventions require careful consideration during tokenizer training.

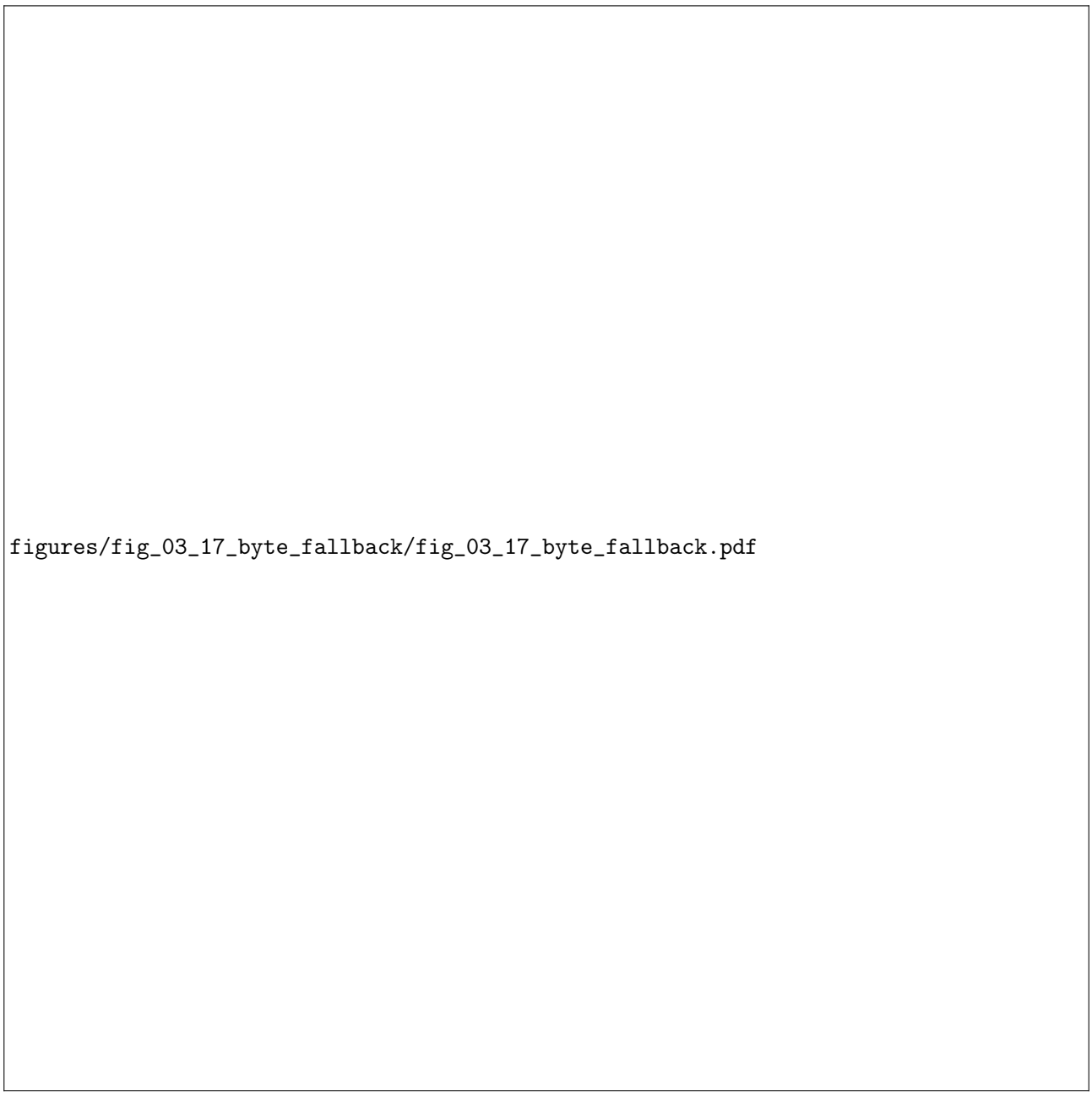

Figure 3.17: When a character is not in the learned vocabulary, byte-level fallback represents it as a sequence of byte tokens corresponding to its UTF-8 encoding. This guarantees that any valid Unicode input can be tokenized without loss of information.

Figure 3.18: Compression ratio (bytes per token) increases with vocabulary size as more text is represented by single tokens rather than character sequences. The relationship between vocabulary size and compression ratio follows diminishing returns.

3.7 Context Representation in Tokenized Language

How This Chapter Represents Context
The fundamental question in language modeling is: How do we represent the context $w_1, \ldots, w_{t-1}$ to predict w_t?

- **Context representation**: Tokenization converts raw text into a sequence of discrete token IDs from a fixed vocabulary $\mathcal{V}$
- **Context encoding**: Each token receives a unique integer index, creating sparse one-hot representations that subsequent layers transform
- **Limitation**: Subword tokens may split meaningful units, requiring models to learn to compose meaning across token boundaries
- **Next chapter preview**: Word embeddings will transform these sparse token IDs into dense vector representations that capture semantic similarity

Tokenization fundamentally shapes how language models represent context for next-token prediction. When we compute $P(w_t \mid w_1, \ldots, w_{t-1})$, the sequence $w_1, \ldots, w_{t-1}$ is not raw text but a sequence of token IDs produced by the tokenizer, creating a discrete symbolic representation of the continuous stream of language. Each token ID is an integer that indexes into the vocabulary $\mathcal{V}$, and the model's first layer converts these integers into dense vector representations through an embedding lookup. The choice of tokenization determines what units compose this context: a word-level tokenizer might represent "The cat sat" as three token IDs, while a BPE tokenizer might produce four or five IDs depending on how it segments each word. The model sees only these token IDs and must learn to compose meaning from whatever units the tokenizer provides, without access to the original character-level text. This means that the tokenizer's decisions about where to place boundaries become baked into the model's understanding of language structure, influencing everything from how the model represents concepts to how it handles novel words.

The implications for prediction are profound, affecting how uncertainty is distributed across token positions. Consider predicting the next token after "un" in isolation versus after "unhappi". In the first case, the model must consider all words beginning with "un"—a vast space including "under", "until", "unusual", and thousands more possible continuations. In the second case, the likely continuations are much more constrained: "ness", "ly", "er". The tokenizer's decision to split "unhappiness" at particular boundaries affects how much information about the eventual word is available to the model at each prediction step. More generally, subword tokenization creates a kind of hierarchical structure where the model first predicts coarse-grained units (common subwords, whole words) and then refines within those units (completions of rare words). This hierarchical structure emerges automatically from the tokenization without being explicitly designed into the model architecture, allowing the model to allocate prediction capacity efficiently across common and rare word forms.

One subtle consequence of subword tokenization is that the model's notion of "word" becomes fuzzy. When predicting after "The capital of France is", a word-level model clearly predicts a single token "Paris". A subword model might predict "Par" followed by "is", or might have learned "Paris" as a single token depending on its frequency in training data. The probability $P(\text{Paris})$ is no longer a simple lookup but might be computed as $P(\text{Par}) \times P(\text{is} \mid \text{Par})$ or accessed directly as $P(\text{Paris})$ depending on tokenization. This makes comparing probabilities across tokenizers subtle: a model's reported perplexity depends on how many tokens it must predict, which varies with tokenization choices. Researchers must be careful to use consistent tokenization when comparing models or to normalize metrics appropriately. The next chapter will explore how embedding layers transform these discrete token IDs into continuous vector representations, providing the foundation for neural language models to learn rich semantic relationships between tokens.

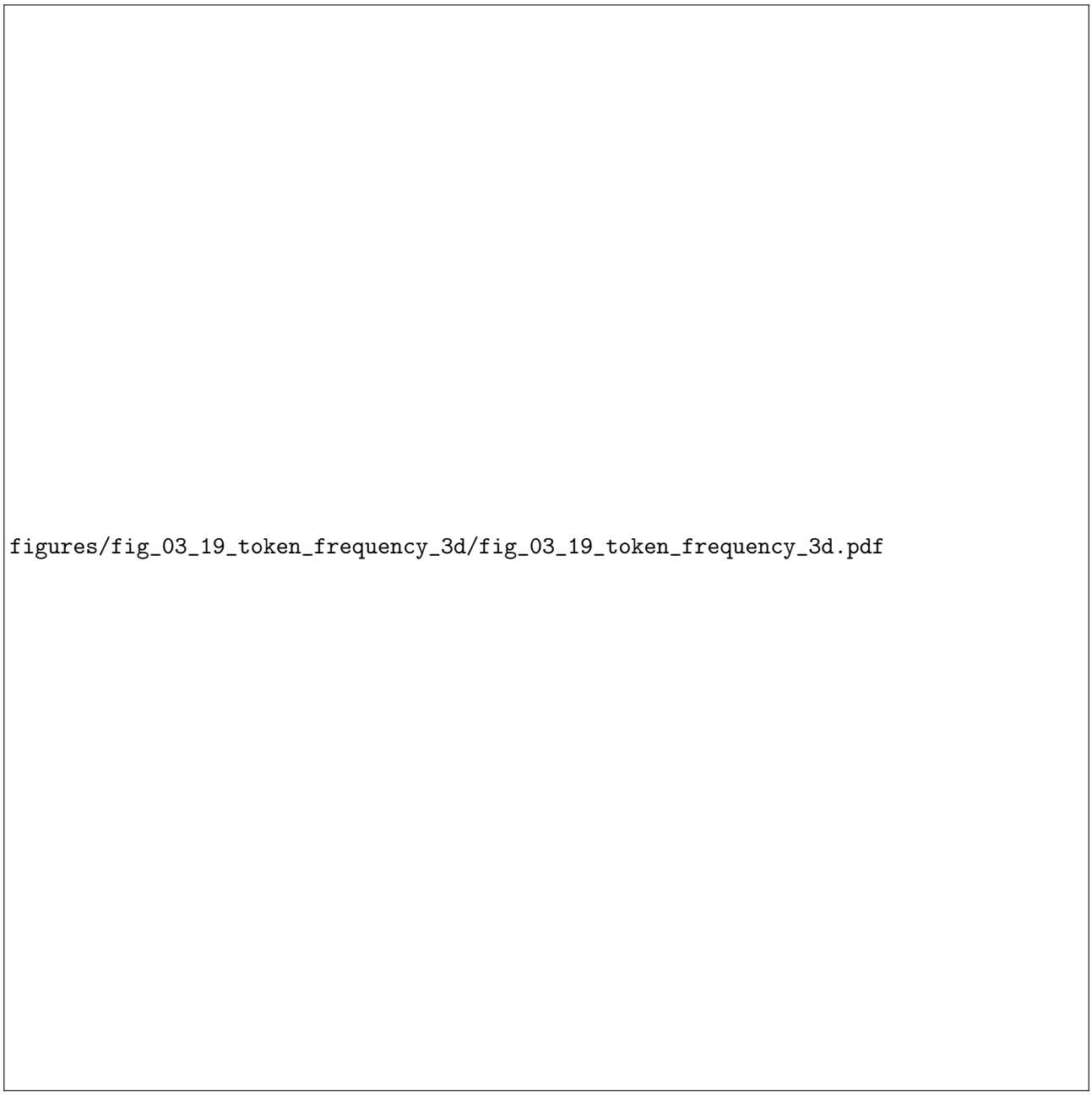

Figure 3.19: A three-dimensional view of token frequencies in a trained vocabulary shows the heavy-tailed distribution: a small number of tokens account for most occurrences, while the majority of vocabulary entries are used rarely.

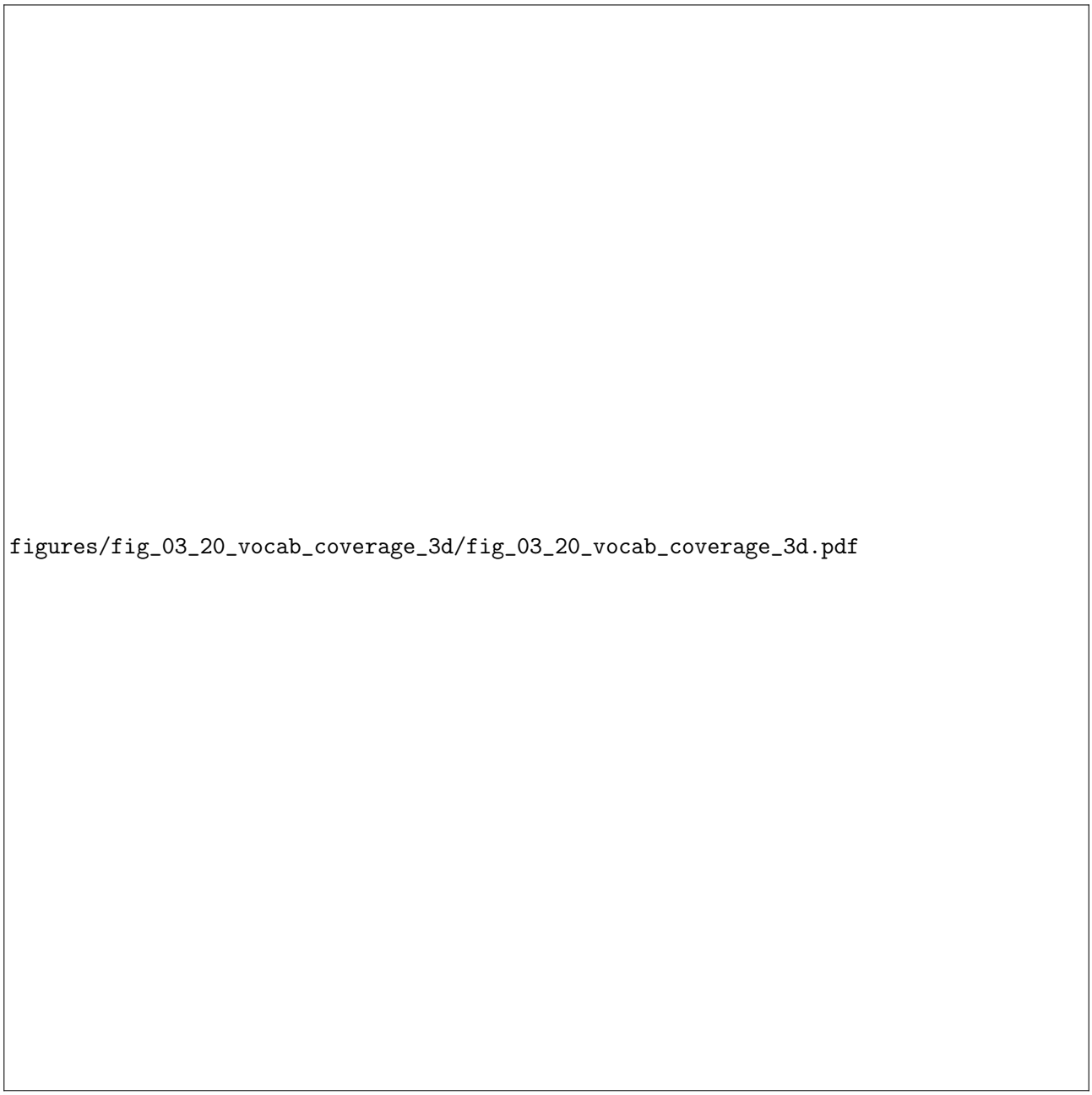

Figure 3.20: A three-dimensional surface showing how vocabulary coverage varies with vocabulary size and language characteristics reveals the diminishing returns of larger vocabularies and the significant differences between languages.

Figure 3.21: The dynamics of BPE merging visualized as a three-dimensional surface shows how vocabulary composition changes as more merge operations are performed, transitioning from character-dominated to subword-dominated to word-dominated regions.

Figure 3.22: A three-dimensional comparison of BPE, WordPiece, and Unigram tokenization across vocabulary size, fertility, and downstream performance illustrates the trade-off space that practitioners must navigate.

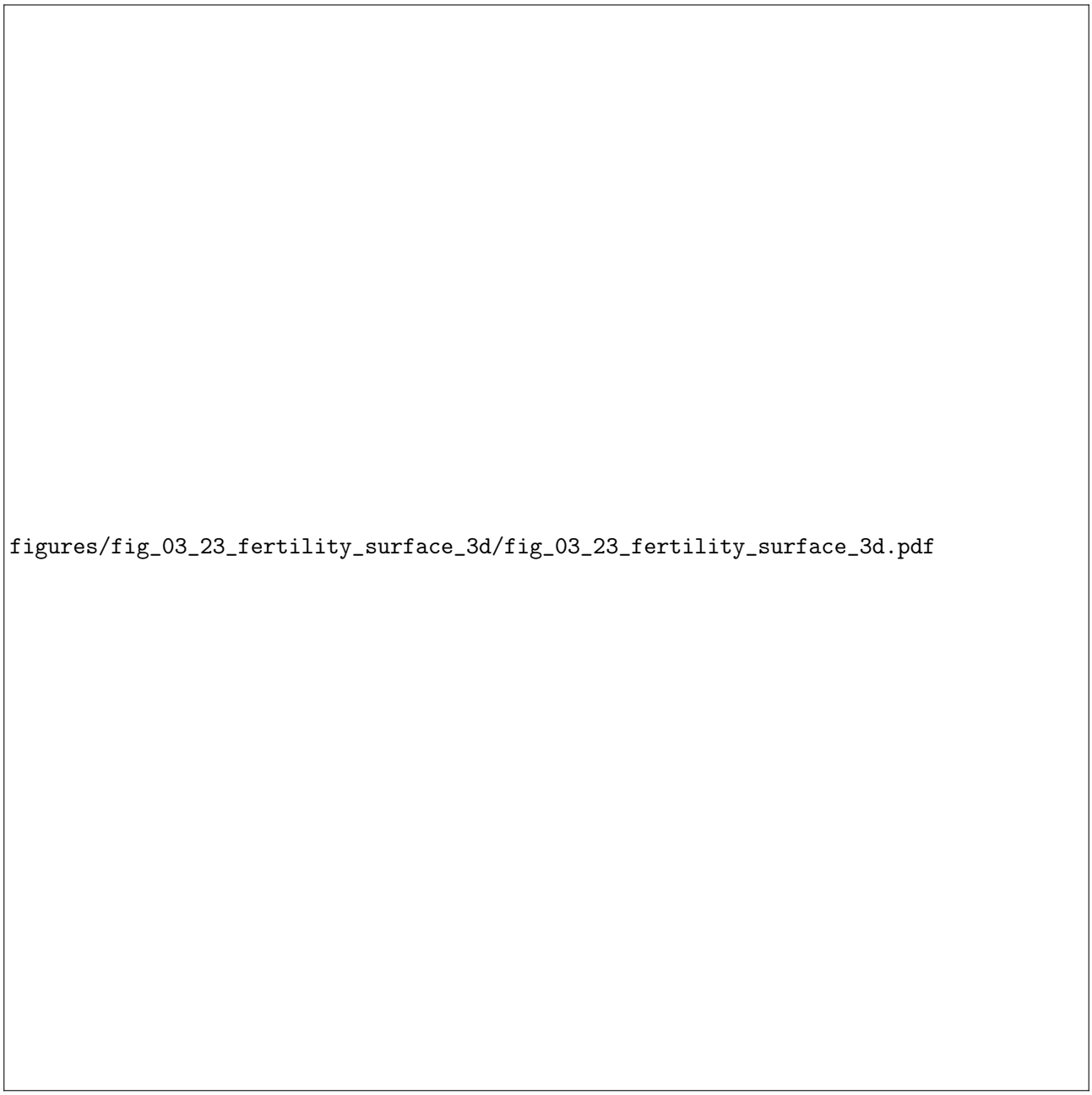

Figure 3.23: Fertility varies as a three-dimensional surface over vocabulary size and language type. The surface reveals how different languages respond to increasing vocabulary size and helps identify optimal vocabulary sizes for multilingual models.

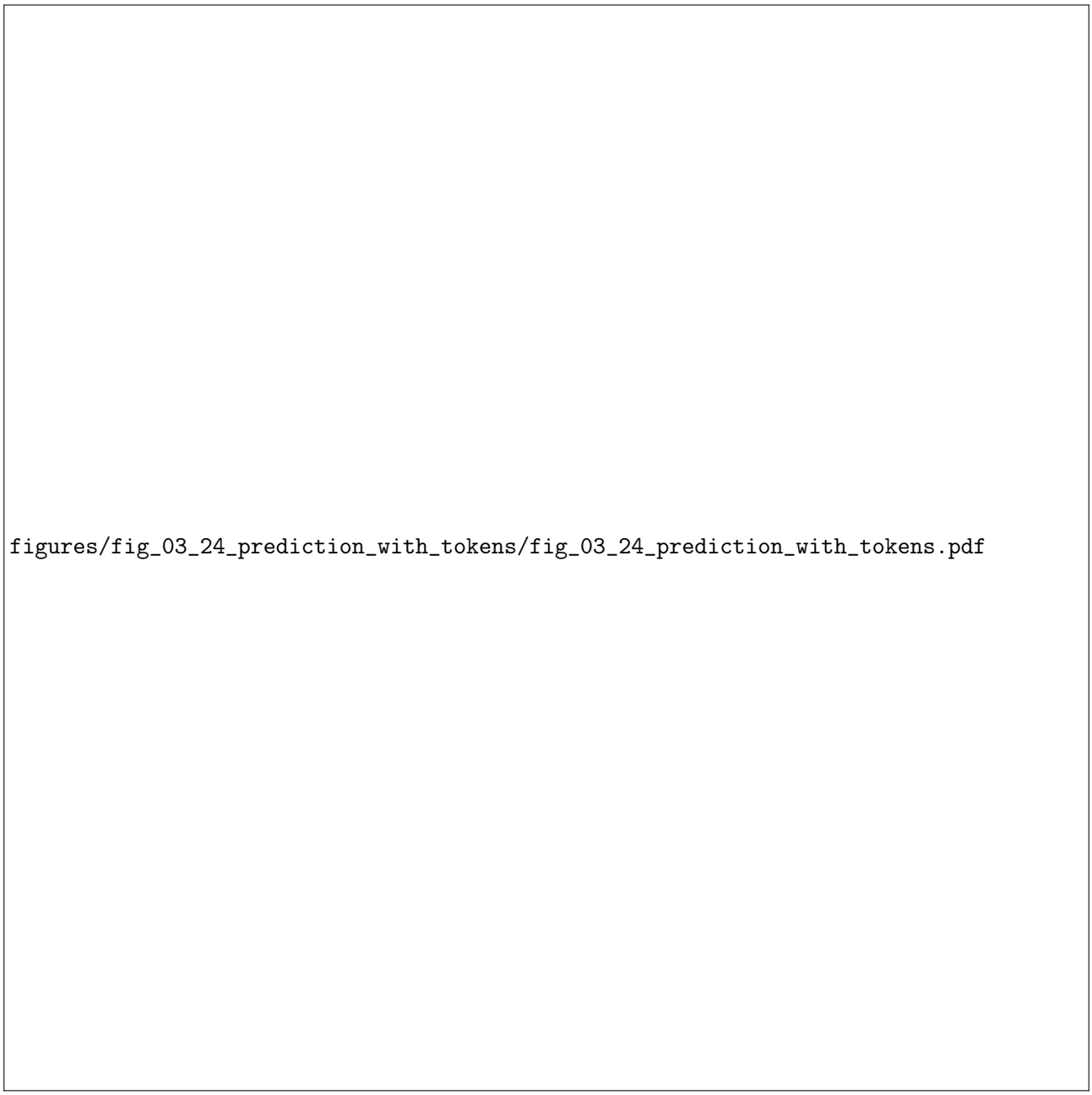

Figure 3.24: Next-token prediction with subword tokenization may require multiple prediction steps to generate a single word. The probability of the complete word is the product of conditional probabilities at each step.

Figure 3.25: The journey from raw text to context representation: tokenization produces token IDs, embedding lookup converts IDs to vectors, and subsequent model layers transform these vectors into rich contextual representations that inform next-token prediction.

3.8 Summary

This chapter has explored tokenization as the critical preprocessing step that defines what units language models predict. We began by examining word-level tokenization, which aligns with human intuition but suffers from out-of-vocabulary problems and vocabulary explosion in morphologically rich languages. Character-level and byte-level approaches solve the coverage problem but create impractically long sequences that strain computational resources and make it harder for models to capture long-range dependencies. Subword tokenization emerged as the dominant solution, with BPE, WordPiece, SentencePiece, and unigram language model methods each offering slightly different approaches to learning vocabularies that balance sequence length against vocabulary size. We examined how vocabulary size affects the embedding matrix, fertility, and downstream performance, and how special tokens provide structural markers beyond simple text representation. The evaluation of tokenization quality through fertility, compression ratio, coverage, and downstream task performance guides practical decisions about tokenizer design. Finally, we explored how tokenization shapes context representation, creating a mapping from raw text to discrete token IDs that subsequent model layers transform into rich contextual representations for prediction.

We can now predict better because:

- Subword tokenization eliminates out-of-vocabulary words while maintaining reasonable vocabulary sizes
- BPE and WordPiece learn data-driven vocabularies that balance word frequency and morphological compositionality
- Byte-level fallback mechanisms ensure complete coverage of any Unicode text, including emoji and rare scripts
- Special tokens provide explicit structure for tasks beyond simple next-word prediction, enabling masked language modeling and sequence classification
- Vocabulary size becomes a tunable hyperparameter that trades off between model capacity and sequence length efficiency

Next: Chapter **??** transforms these discrete token IDs into continuous vector representations, enabling models to capture semantic similarity between words that tokenization treats as completely independent symbols.

Exercises

1. **Word-Level Vocabulary Analysis.** Given a corpus of 1 million words with 50,000 unique word types, calculate what percentage of word types appear fewer than 5 times. Explain why this creates problems for word-level tokenization and how it motivates subword approaches.

2. **BPE Merge Steps.** Starting with the character vocabulary {a, b, c, d, e, s, t, _} and the corpus "abracadabra", perform 3 iterations of the BPE merge algorithm. Show the vocabulary after each merge and explain why each pair was selected.

3. **Vocabulary Size Trade-offs.** A language model uses vocabulary size $|\mathcal{V}| = 32,000$. If we double the vocabulary to 64,000, explain how this affects: (a) the embedding matrix size, (b) average tokens per sentence, and (c) the model's ability to handle rare words.

4. **Fertility Calculation.** Given a tokenizer that produces the following tokenizations: "unhappiness" $\rightarrow$ ["un", "happiness"], "internationalization" $\rightarrow$ ["international", "ization"], calculate the fertility for each word and the average fertility across both examples.

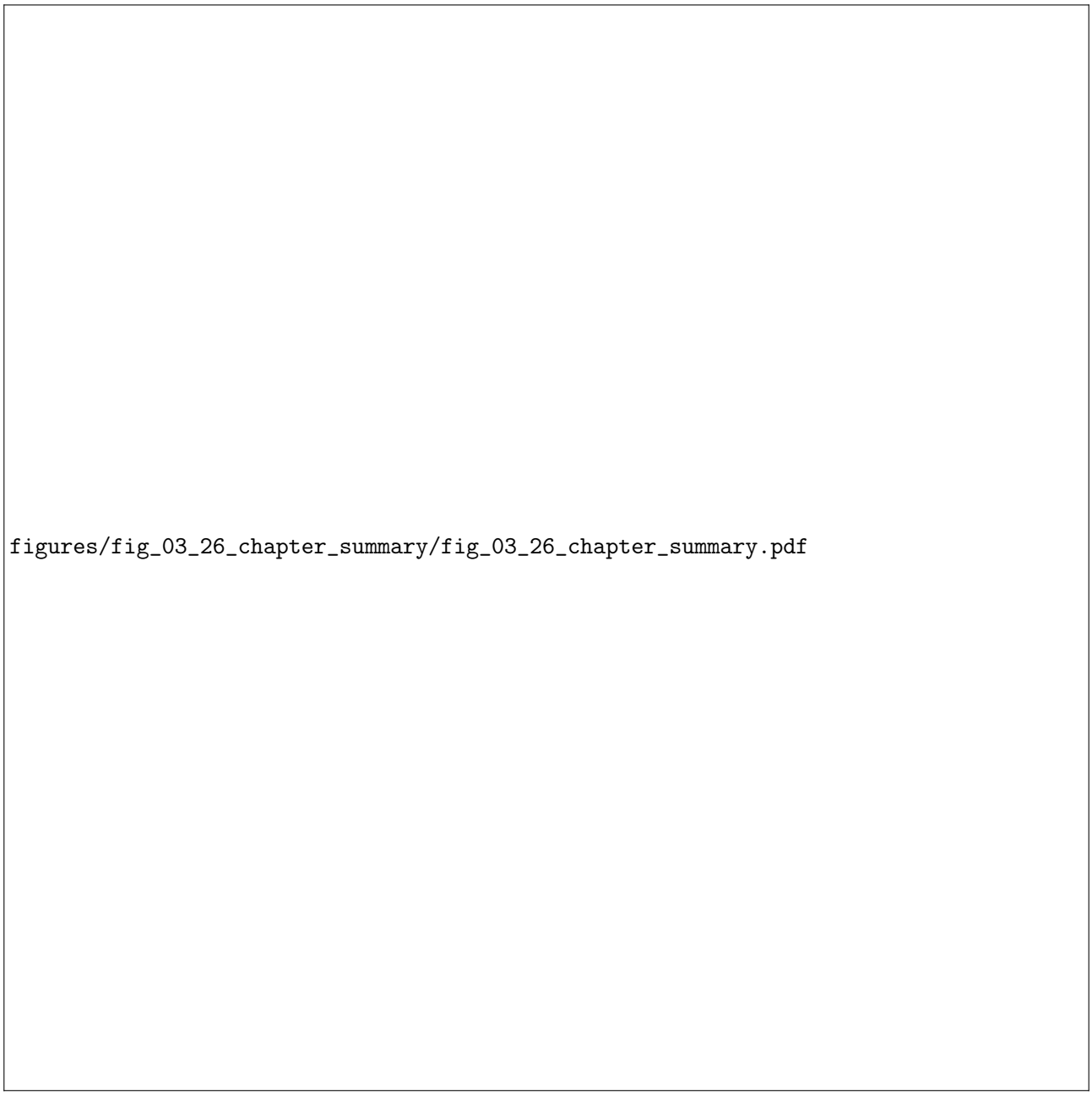

Figure 3.26: Chapter 3 summary: Tokenization converts raw text into discrete token IDs that define the vocabulary $\mathscr{V}$ over which language models predict. Subword algorithms balance vocabulary size against sequence length, while special tokens provide structural markers for tasks beyond simple language modeling.

5. **Special Tokens.** Explain the purpose of each special token: [CLS], [SEP], [MASK], [PAD], [UNK]. For each, describe a specific scenario where it is essential for model training or inference.

6. **Byte-Level Fallback.** The Unicode character for the emoji "heart" has UTF-8 encoding. Explain how a byte-level tokenizer handles this character when it is not in the learned vocabulary, and why this guarantees no out-of-vocabulary tokens.

7. **WordPiece vs BPE.** Compare how BPE and WordPiece would tokenize the word "unbreakable" if both have learned prefixes "un" and "able" but neither has learned "break" as a single token. Explain the key algorithmic difference between the two methods.

8. **Multilingual Tokenization.** A SentencePiece model trained on English text is applied to German text containing the word "Donaudampfschifffahrtsgesellschaft". Predict how this compound word might be tokenized and explain why language-specific training matters for tokenizer quality.

9. **Sequence Length Impact.** A character-level model processes the sentence "The quick brown fox" (19 characters) while a BPE model tokenizes it into 5 tokens. Calculate the ratio of computational cost (assuming self-attention with $O(n^2)$ complexity) and discuss the implications for training efficiency.

10. **Tokenization and Prediction.** Consider predicting the next token after "The capital of France is". Compare how a word-level, character-level, and subword tokenizer would frame this prediction task. Which representation makes the prediction easiest for the model?

11. **Research Question: Optimal Vocabulary Size.** Design an experiment to determine the optimal vocabulary size for a specific language and domain. What metrics would you use to evaluate tokenizer quality, and how would you balance compression ratio against downstream task performance?

Bibliography

Taku Kudo and John Richardson. Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. *arXiv preprint arXiv:1808.06226*, 2018.

Rico Sennrich, Barry Haddow, and Alexandra Birch. Neural machine translation of rare words with subword units. *arXiv preprint arXiv:1508.07909*, 2016.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, et al. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016.

Index