Boudica Torc - BPE Vocabulary

Overview of Vocabulary Types

This document explains the vocabulary creation choices used in modern language models, focusing on the strategies available within the **Boudica Torc** platform. We will cover full-word, character-level, and subword (Byte-Pair Encoding - BPE) vocabularies, outlining their trade-offs and implementation details.

- **Full-word:** Each distinct word (typically delimited by whitespace) is a unique token in the vocabulary. While simple, this approach struggles with rare or unseen words.
- **Character-level:** Each Unicode character is a token. This results in a very small vocabulary and eliminates out-of-vocabulary (OOV) issues, but it creates long token sequences, increasing computational cost.
- **Subword (BPE):** The preferred method for Boudica Torc. It starts with individual characters and iteratively merges the most frequent adjacent pairs into new, multi-character subword tokens. This balances vocabulary size, sequence length, and OOV handling effectively.

Why Vocabulary Choice Matters

The selection of a vocabulary strategy has a direct impact on model performance, memory usage, and computational requirements.

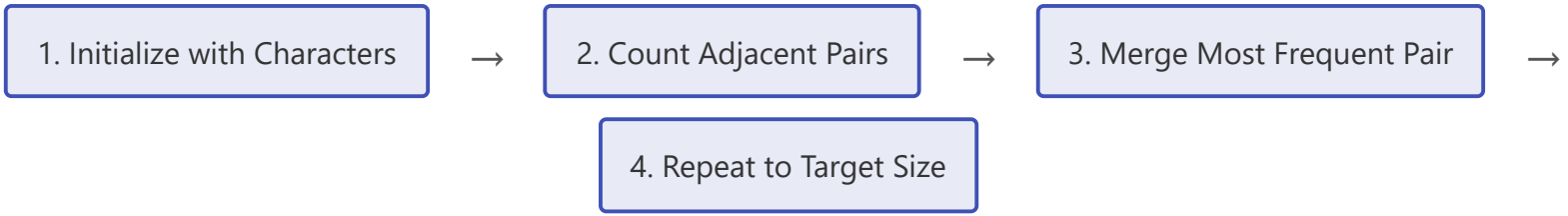
- **Embedding Table Size:** The memory required is calculated as `vocab_size × embedding_dim`. A larger vocabulary directly increases the model's memory footprint and checkpoint size.

- **Sequence Length:** Vocabularies with smaller tokens (like character-level) produce longer sequences for the same text, which increases compute time during both training and inference.
- **Out-of-Vocabulary (OOV) Handling:** Full-word models fail on unseen words, requiring a generic token. BPE and character-level models can represent any word, making them more robust.
- **Morphology:** For languages with rich morphology (e.g., German, Turkish), subword and character vocabularies are superior as they can break down complex words into meaningful stems, prefixes, and suffixes.

Understanding Byte-Pair Encoding (BPE)

BPE provides a practical middle ground by creating a vocabulary of frequently occurring subwords. This allows the model to handle known words efficiently while still being able to construct rare or new words from smaller, known pieces.

Conceptual BPE Merge Process



Vocabulary Trade-offs: A Comparison

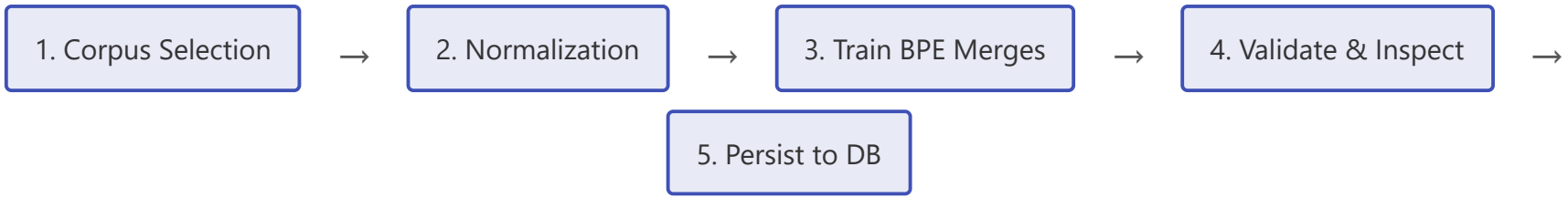
Vocabulary Type	Pros	Cons
Full-word	- Intuitive tokens - Short sequences for known words	- Very large vocabulary required for good coverage - Fails on out-of-vocabulary (OOV) words - Large embedding table

Vocabulary Type	Pros	Cons
Character	- Minimal vocabulary size - No OOV words - Robust to misspellings and noise	- Very long token sequences - High computational cost - Harder for models to learn composition
BPE / Subword	- Best practical trade-off - Manages rare words gracefully - Smaller vocabulary than full-word - Shorter sequences than character-level	- Requires a corpus-specific training step - Tokens are less human-interpretable than full words

Boudica Torc: BPE Pipeline Implementation

Boudica Torc uses a configurable and reproducible BPE subword pipeline as the default tokenization strategy. This pipeline is fully integrated with the repository's database schema, ensuring all tokenizer artifacts are versioned, discoverable, and auditable.

High-Level Pipeline



Implementation and Code-Level Details

The core BPE pipeline is implemented in C++ (`src/tokenizer.cpp`) and offers a reproducible and scalable solution.

- **Performance:** The implementation uses OpenMP for parallelization and supports optional GPU acceleration (`src/tokenizer_gpu.cu`) for rapid processing of large corpora.
- **End-of-Word Marker:** An explicit marker is appended to the last character of each word. This preserves word boundaries during the merging process, which aids in clean detokenization.
- **Training CLI:** The vocabulary can be built using the included `slm_train` command-line helper. The vocabulary size is sourced from the model's configuration file.

```
# Build vocabulary from a newline-delimited corpus file
./slm_train --build-vocab corpus.txt

# Programmatic usage (C++):
# include "src/tokenizer.hpp"
boudica::Tokenizer::build_vocabulary(texts, vocab_size);
```

Database Integration and Persistence

A key feature of the Boudica Torc platform is the tight integration of tokenizer artifacts with the model registry and database schema. This ensures reproducibility and operational consistency.

1. **Vocabulary Storage:** The final token-to-ID mapping and frequency counts are persisted in the `boudislm.vocabulary` table.
2. **Model Registry:** Key metadata such as `vocab_size`, tokenizer type, and configuration details are stored in `boudislm.model_registry` for each checkpoint.
3. **Embeddings:** Learned token vectors are stored in `boudislm.token_embeddings`, keyed by `token_id`.

Operational Best Practices and Recommendations

- **Default Approach:** Use a subword method (BPE or SentencePiece) by default. It provides the best balance of performance, memory, and robustness for production deployments.
- **Vocabulary Size:** Target an initial size of **32k–50k** for English-dominant models. Expand to **64k+** for multilingual or domain-rich corpora.
- **Reproducibility:** Always version control tokenizer configurations, merge lists, and the corpus snapshot hash alongside model checkpoints. Store this information in the `boudislm.model_registry.architecture` field.
- **Normalization:** Use Unicode-aware BPE with consistent normalization (e.g., NFC) unless you expect to handle mixed-encoding or binary-like inputs, in which case byte-level BPE is preferable.

Appendix: Vocabulary Creation Checklist

1. Choose tokenization normalization rules (case, punctuation, Unicode form).
2. Decide on the base unit (bytes vs. characters).
3. Preprocess the training corpus consistently (cleaning, deduplication, filtering).
4. Train the BPE merges to the target vocabulary size and save the resulting artifacts (merge list, vocab file).
5. Insert special tokens (e.g., , ,) and reserve their IDs.
6. Update the database: populate `boudislm.vocabulary` and initialize placeholder rows in `boudislm.token_embeddings`.
7. Record the final `vocab_size` and all tokenizer metadata in `boudislm.model_registry` for the associated model checkpoint.

Created by Boudica February 06 2026