Dictionary-Based Speculative Decoding for Non-Latin-Script Languages

Oleksiy Syvokon
Lviv Polytechnic National University
oleksii.o.syvokon@lpnu.ua

Abstract

Large language models tokenize non-Latin-script languages inefficiently: a single word in Ukrainian or Crimean Tatar is split into two to three times as many tokens as its English equivalent. We propose *dictionary-based speculative decoding* (DictSpec), which accelerates inference by proposing draft continuations from a static n-gram lookup table built offline from an unlabeled corpus. The lookup table requires no trainable parameters or GPU resources, is inexpensive to construct, adds under 5 MB of memory overhead, and can be reused across models that share a tokenizer. We evaluate DictSpec on Ukrainian and Crimean Tatar (Cyrillic and Latin scripts), implementing a vLLM plugin to benchmark five models ranging from 3B to 70B parameters on consumer- and server-grade GPUs. In controlled emulation, DictSpec reduces verification steps by up to 1.65×, with gains correlating substantially with tokenizer fertility. In live vLLM serving, pure DictSpec gives modest speedups, while a hybrid with prompt-local n-gram speculation reaches up to 1.76×. We release our code and vLLM plugin as open source.[1]

1 Introduction

Inference cost for large language models is not uniform across languages. Tokenizers trained on English-dominated corpora assign most common English words a single token but fragment words in non-Latin scripts into many subword pieces. Each piece requires a separate forward pass through the model. For languages such as Ukrainian, Georgian, Hindi, and Crimean Tatar, this creates a *tokenization tax* that makes inference substantially slower (Rust et al., 2021). We target this latency gap without modifying the model or tokenizer.

Our main insight is that this inefficiency is not uniformly distributed across decoding steps. The model faces a genuinely difficult decision when it must choose which word or phrase to begin. But once the initial tokens of a word have been committed, the remaining tokens are often highly predictable. A model that has generated the Ukrainian token prefix "`_пер·сон·аль·ний _ком`" almost certainly intends to continue with "`п'·ют·ер`", completing *"персональний комп'ютер"* ("personal computer").

Standard LLM inference spends the same compute on a predictable token as on a genuinely hard one. *Speculative decoding* exploits this gap: a cheap draft mechanism proposes a multi-token continuation, and the target model verifies the entire proposal in a single forward pass, accepting correct tokens and falling back on mismatches.

We propose *dictionary-based speculative decoding* (DictSpec), which realizes this idea with a simple draft mechanism: a static lookup table built offline from a corpus in the target language. The table maps short token-sequence prefixes to their most probable continuations. For instance, the table maps the word prefix "*головнок*" ("commander-i…") to the continuation completing "*головнокомандувач*" ("commander-in-chief"). When more preceding context is available, even shorter prefixes become predictable: the tokens for "*верховний го*" ("supreme commander-i…") are enough to predict the full word "*головнокомандувач*", because "*верховний*" ("supreme") strongly disambiguates what follows. At inference time, the method checks the tail of the tokens generated so far against the lookup table. If a matching prefix is found, the stored continuation is proposed as a draft. The target model then verifies the draft using the standard speculative decoding protocol (Leviathan et al., 2023; Chen et al., 2023). If the proposed tokens are accepted, multiple decoding

[1] https://github.com/osyvokon/dictspec

Proceedings of the Fifth Ukrainian Natural Language Processing Conference (UNLP 2026), pages 169–183

steps are resolved in a single verification pass. If any proposed token is rejected, decoding falls back to standard autoregressive generation from that point. Because we use the standard speculative decoding verification, the output distribution is provably identical to that of unmodified autoregressive decoding.

Building the lookup table requires only an unlabeled monolingual text corpus, takes a few minutes on a consumer-grade laptop without a GPU, and produces a compact data structure requiring under 5 MB of CPU memory. Because a dictionary is tied to a tokenizer and language/script rather than to model weights, a single table serves every model in a family and can be reused without modification across inputs and sessions.

We evaluate the method across seven checkpoints from five tokenizer families (Gemma 3, Mistral, Mistral NeMo, Qwen 3.5, and Llama 3) in a controlled emulation study over Ukrainian and Crimean Tatar in both Cyrillic and Latin scripts, sweeping dictionary construction parameters across 1,350 total configurations, and validate with wall-clock benchmarks in vLLM.

Our main contributions are:

- We propose DictSpec, a simple and lightweight draft mechanism that accelerates LLM inference for non-Latin-script languages while preserving the target model's output distribution exactly.
- We find that speedup correlates substantially with tokenizer fertility (Pearson $r = 0.72$). The method helps most where tokenization is least efficient.
- We integrate the method into vLLM and evaluate pure DictSpec, prompt n-gram speculation, and a hybrid of the two. Pure DictSpec yields modest positive wall-clock gains on coherent outputs, while the hybrid reaches up to 1.76×.

2 Related Work

2.1 Tokenization Inefficiency in Non-Latin Scripts

Subword tokenization methods such as Byte-Pair Encoding (Sennrich et al., 2016) and SentencePiece (Kudo and Richardson, 2018) build vocabularies that reflect the statistical distribution of the training corpus. When the corpus is dominated by English and other high-resource Latin-script languages, the resulting tokenizer allocates most vocabulary entries to those languages and offers poor coverage of non-Latin scripts. Rust et al. (Rust et al., 2021) showed that tokenizer quality varies dramatically across languages, and that poor tokenizers inflate sequence length and degrade model performance for morphologically rich or non-Latin-script languages. This over-segmentation increases both memory consumption and latency, since each additional token requires a separate autoregressive forward pass. Recent work has documented similar findings for Ukrainian specifically (Maksymenko and Turuta, 2025; Kiulian et al., 2024; Kiulian et al., 2025), and broader studies have shown that tokenizer fragmentation introduces systematic cost and latency disparities across languages (Petrov et al., 2023; Ahia et al., 2023; Hong et al., 2024).

2.2 Extending or Retraining Tokenizers

One line of work addresses tokenization inefficiency by training a new tokenizer or extending the existing vocabulary to better cover the target language (Minixhofer et al., 2022; Cui et al., 2023; Kiulian et al., 2025). Lapa LLM reports replacing a large portion of the base Gemma vocabulary with Ukrainian-specific tokens in order to reduce tokenization cost and improve efficiency for Ukrainian generation (Paniv et al., 2025). Such methods can substantially reduce tokenizer fertility and improve generation quality for underrepresented languages, but they require re-embedding the new vocabulary tokens and continued pretraining on the target language. Moreover, this adaptation must be repeated for every model architecture and size, which makes it a costly process that does not transfer across models. They may also degrade performance on non-target languages by redistributing vocabulary capacity. Our method requires no modification of the model or its tokenizer, and the output distribution remains exactly that of the original model.

2.3 Speculative Decoding

Neural draft models. The idea of verifying multiple predicted tokens in parallel was proposed by (Stern et al., 2018). Speculative decoding was later formalized with distribution-preserving guarantees by (Leviathan et al., 2023) and (Chen et al., 2023). The original formulation uses a smaller language model as the draft model; the target model verifies the proposed block in a single forward pass, accepting or rejecting tokens according

	Step	Example
1.	Extracted n-grams	`персональний комп'ютер (count=12,400)` `персональний комунікатор (count=180)`
2.	Tokenized forms	`_пер сон аль ний _ком п' ют ер;` `_пер сон аль ний _ком уні ка тор`
3.	Context prefix	`_пер сон аль ний _ком`
4.	Candidate continuations	`п' ют ер (prob=98.6%; selected)` `уні ка тор (prob=1.4%; discarded)`
5.	Lookup entry (text)	`_пер сон аль ний _ком → п' ют ер`
6.	Lookup entry (ids)	`[12, 47, 91, 103, 255] → [301, 404, 509]`

Table 1: Schematic illustration of the dictionary construction procedure for the Ukrainian n-gram `персональний комп'ютер` ("personal computer"). Spaces mark token boundaries; _ denotes a token with a leading space.

to a rule that preserves its marginal distribution. A practical limitation is that a suitable draft model must be obtained for each target model family and language: a good English draft model may be a poor draft model for Ukrainian, and training one for a new language requires additional training data and GPU resources.

Architectural methods. A second line of work embeds draft generation into the target model itself. Medusa (Cai et al., 2024) adds multiple decoding heads, each predicting a different future token; EAGLE (Li et al., 2024) builds a lightweight draft network from the target model's hidden states. Both achieve strong speedups but require architecture modifications and additional training for the target model. Most relevant to our motivation, Hong et al. (Hong et al., 2024) target the speed penalty caused by excessive tokenization of non-Latin scripts with a language-specific decoding head (reporting 1.7× speedup), but their method still requires training the new head for each target language and model.

Non-neural methods. The approach closest to ours replaces the neural draft model with n-grams drawn from the prompt or previously generated text (Saxena, 2023). Prompt n-grams work well when output closely resembles input (e.g., document editing or summarization) but offer lower coverage in open-ended generation. REST (He et al., 2024) broadens coverage by retrieving continuations from a datastore. More recent work builds n-gram tries from the input context for in-context learning (Chen et al., 2025), and Stewart et al. (Stewart et al., 2024) show that learning-free n-gram statistics alone can yield competitive speedups. Our method differs in that the lookup table is built offline from a large corpus, giving high coverage of common continuations in the target language even when the prompt provides little relevant context. The two strategies are complementary: prompt n-grams exploit repetition within the current context, whereas DictSpec exploits corpus-level regularities of the target language. We evaluate both methods separately and in combination in our vLLM experiments.

3 Method

3.1 Dictionary Construction

Our method builds a lightweight lookup table that maps token prefixes to their most probable continuations. Building the table requires only an unlabeled text corpus in the target language, is performed entirely offline, and needs no GPU or model access. Table 1 illustrates the full procedure on a concrete example.

We first extract frequent words and short n-grams with their counts (Step 1) and tokenize each using the target model's tokenizer (Step 2). From the resulting token sequences we build prefix-to-continuation statistics: for each shared token prefix (Step 3) we record which continuation is most likely (Step 4) and store only that single most probable suffix (Steps 5–6). Unlike a standard count-based n-gram language model that estimates the full next-token distribution, our table keeps only the top continuation. This approach, combined with an efficient trie (Yata, 2023) implementation, allows us to fit the whole lookup table in just 2 to 5 MB, with median lookup latency below 7 µs.

To keep the table compact and improve acceptance probability, we apply two filters. First, we

retain only the most frequent n-grams. Second, we apply a minimum probability threshold so that we do not store ambiguous continuations, which are likely to be rejected during verification. The resulting dictionary is tied to the tokenizer and language/script, so it can be reused across models in a family. In our setting, building it takes a few minutes on a consumer-grade laptop.

3.2 Speculative Decoding

Speculative decoding accelerates autoregressive generation by having a cheap draft mechanism propose a short continuation of up to γ tokens (the *speculative budget*) that the target model then verifies in a single forward pass. The target model checks the proposal left to right: accepted tokens are appended and generation advances by several positions at the cost of one verification step; when a mismatch occurs, verification stops and the target model supplies the correct next token. Poor proposals reduce efficiency but never compromise correctness. In the worst case the method degrades to standard autoregressive decoding, and the standard verification procedure preserves the exact output distribution of the target model (Leviathan et al., 2023).

3.3 Dictionary-Based Speculative Decoding

The target model must still decide which word or phrase to begin, but once the first part of a word has been generated, the rest is often predictable. Our method uses the dictionary to propose that predictable continuation. The target model then verifies it.

At each decoding step we search the dictionary for the longest suffix of the current token history that appears as a key, backing off to shorter contexts only if necessary. This longest-match rule yields the most specific completion. For instance, referring to Table 1, after the model has generated tokens `_пер сон аль ний _ком`, the dictionary matches the full five-token prefix and proposes the continuation `п' ют ер` (Step 5). The stored continuation is verified by the target model. If the model agrees, all three tokens are accepted in one step; if it instead prefers "комунікатор", verification stops at the first mismatch and the model supplies the correct token. The example also shows why longer context matters: `_ком` alone is ambiguous, whereas the full prefix is specific. When no match is found, DictSpec emits no draft, and decoding proceeds with a standard model step.

Because the dictionary operates over token IDs, it can match partial sequences inside words, not only at word boundaries, so speculation can begin as soon as the recent suffix becomes distinctive.

The dictionary and prompt-local n-gram speculation provide complementary draft sources. DictSpec can propose common language-level continuations that have not appeared in the current prompt, while prompt n-grams can exploit repetition in the input or generated context. In the vLLM experiments we therefore evaluate not only pure DictSpec, but also a hybrid mode: the proposer first queries the corpus dictionary and, when no dictionary draft is available, falls back to vLLM's prompt n-gram proposer. Both sources are still verified by the target model under the standard speculative decoding protocol, so the output distribution is unchanged.

4 Experiments

We evaluate in two stages. First, we run a controlled emulation (Section 4.5) that isolates the speculative decoding mechanism from system-level variables by replaying pre-tokenized reference text. Second, a wall-clock validation with vLLM (Section 4.6) confirms that the emulated gains translate to real throughput improvements in a production serving engine.

4.1 Models

We evaluate seven recent checkpoints from five tokenizer families: Gemma 3 (Gemma Team, 2025), Mistral (Jiang et al., 2023), Mistral NeMo, Qwen 3.5 (Team, 2026), and Llama 3 (Grattafiori et al., 2024). Because each dictionary is tied to the tokenizer, models within a family produce identical emulation results; we therefore report one row per family in the emulation study. For wall-clock validation we select models at two scales: 3–4B on a consumer GPU and 27–70B on a server GPU (Section 4.6). We include the older Mistral 7B v0.1 to assess whether tokenizer quality has improved over three years.

4.2 Languages and Datasets

We focus on settings where tokenization inefficiency is especially relevant: Ukrainian as a mid-resource language using Cyrillic script, and Crimean Tatar in Cyrillic and Latin scripts as a low-resource language. The Latin-script Crimean Tatar setting serves as a contrastive low-resource condition.

Language		Train words	Valid words
Ukrainian		251,618,586	124,703
Crimean (Cyrillic)	Tatar	1,333,103	535,156
Crimean (Latin)	Tatar	118,749	16,020

Table 2: Dataset statistics for dictionary construction (train) and evaluation (valid).

Model	en	uk	cr-cyr	cr-lat
Gemma 3	1.28	2.38	3.30	2.98
Mistral	1.42	3.18	4.20	3.92
Mistral NeMo	1.26	2.62	3.51	3.07
Qwen 3.5	1.30	2.59	3.51	3.01
Llama 3.3	1.23	2.38	3.89	2.94

Table 3: Tokenizer fertility (average tokens per word) for each model–language pair. Higher values indicate more fragmented tokenization.

For Ukrainian we build the dictionary from a sample of CulturaX (Nguyen et al., 2024). For Crimean Tatar we use the QIRIM Crimean Tatar monocorpus (QIRI'M Young, 2024), split by script. In all cases the training-side corpus is used only for offline dictionary construction and never for target-model adaptation. Table 2 summarizes the dataset sizes.

4.3 Tokenizer Fertility

We quantify tokenization inefficiency using *tokenizer fertility*: the average number of tokens per whitespace-delimited word. We compute fertility on the parallel sentences of FLORES+ (NLLB Team et al., 2024) for English, Ukrainian, and Crimean Tatar Latin. For Crimean Tatar Cyrillic, which is absent from FLORES+, we use the QIRIM validation split.

As Table 3 shows, English fertility is uniformly low (1.2–1.4 tokens per word), while Crimean Tatar Cyrillic is the most fragmented (3.3–4.2), followed by Crimean Tatar Latin (2.9–3.9) and Ukrainian (2.4–3.2). Newer tokenizers reduce fragmentation but still exceed 3 tokens per word on Crimean Tatar Cyrillic.

4.4 Dictionary Construction

The offline dictionary is built from frequent words and short n-grams extracted from the training corpus, tokenized with the target model's tokenizer. For each prefix we retain only the most probable continuation above a minimum probability threshold, truncating both prefixes and completions to at most 8 tokens.

We sweep three dictionary-construction parameters:

- N-gram order: unigrams (`n1`), up to bigrams (`n12`), up to trigrams (`n123`).
- Dictionary size: 10k, 50k, 100k, 200k, 500k, 1M entries.
- Minimum continuation probability: 0.2, 0.5, 0.8, 0.9, 1.0.

4.5 Controlled Emulation Study

We first evaluate with an exact speculative-decoding emulator that replays pre-tokenized reference text, simulating the draft-then-verify cycle under identical conditions across all configurations. This isolates the speculative mechanism from factors such as batching, memory bandwidth, and GPU kernel implementation, and is cheap to run and repeat. Because the reference text is drawn from a static corpus rather than generated by a model, acceptance rates may differ from those observed during real inference; we address this gap with wall-clock experiments on live models in Section 4.6.

At each step the emulator queries the dictionary for the longest matching suffix (up to 8 tokens of context) and proposes a continuation of up to $\gamma = 8$ tokens. The sweep covers 5 tokenizer families, 3 language/script settings, 3 n-gram orders, 6 dictionary sizes, and 5 probability thresholds, for a total of 1,350 runs.

4.5.1 Metrics

Emulated speedup. The ratio of total tokens generated to the number of target-model verification steps, i.e. $\frac{N_{\text{tokens}}}{N_{\text{steps}}}$. Under standard autoregressive decoding every token requires one step, so this ratio is 1; values above 1 indicate that speculative drafting has reduced the number of required forward passes. This metric provides a hardware-independent measure of speculative efficiency.

Draft coverage. The fraction of decoding steps where the dictionary proposes a non-empty draft.

Lang	Speedup	Coverage	MAL	Accept
Gemma 3				
uk	1.26	0.59	0.44	0.17
cr-cyr	1.45	0.71	0.63	0.18
cr-lat	1.19	0.57	0.33	0.10
Mistral 7B				
uk	1.43	0.70	0.61	0.17
cr-cyr	1.65	0.77	0.84	0.18
cr-lat	1.35	0.69	0.50	0.08
Mistral NeMo				
uk	1.34	0.64	0.53	0.18
cr-cyr	1.51	0.73	0.70	0.18
cr-lat	1.21	0.61	0.34	0.07
Qwen 3.5				
uk	1.34	0.63	0.54	0.18
cr-cyr	1.51	0.72	0.71	0.19
cr-lat	1.21	0.58	0.36	0.11
Llama 3.3				
uk	1.28	0.61	0.46	0.18
cr-cyr	1.61	0.77	0.79	0.18
cr-lat	1.22	0.62	0.36	0.07

Table 4: Best emulated speculative speedup for each model–language pair (best configuration across all sweep dimensions). Coverage = draft coverage, MAL = mean accepted draft length, Accept = token-level draft acceptance rate.

Mean accepted draft length (MAL). The average number of drafted tokens accepted per draft-producing step.

Acceptance rate. The fraction of all individually proposed draft tokens that are accepted by the target model.

4.5.2 Main Results

We first ask whether the method reduces decoding work at all. Table 4 reports the best achieved speedup for each model–language pair in the sweep. Speedups are highest for Crimean Tatar Cyrillic (up to 1.65×), which also has the highest tokenizer fertility, and lowest for Crimean Tatar Latin (1.19–1.35×).

The per-token acceptance rate is modest (0.07–0.19), but this does not prevent meaningful speedups. Because the speculative budget allows drafts of up to 8 tokens, even a draft where only the first 1–2 tokens are accepted still advances generation by 2–3 tokens per verification step (number of accepted tokens plus one "bonus" token).

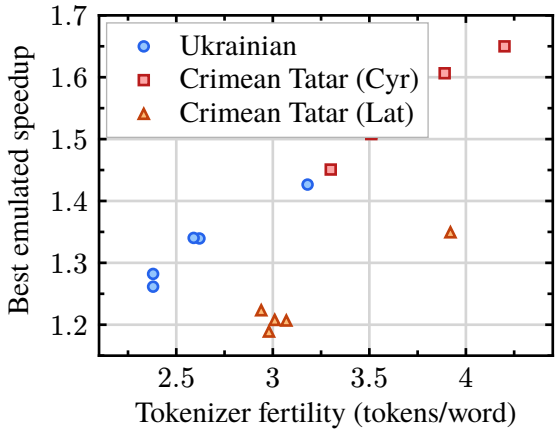

Figure 1: Tokenizer fertility versus best emulated speculative speedup. Higher fertility correlates with larger gains from dictionary-based speculation.

4.5.3 Language and Tokenizer Fertility

Figure 1 plots tokenizer fertility against best achieved speedup for every model–language pair. The correlation is substantial and statistically significant (Pearson $r = 0.72$, $p = 0.002$): Crimean Tatar Cyrillic, the most fragmented setting, consistently achieves the highest speedups. The one outlier is Crimean Tatar Latin, which has higher fertility than Ukrainian yet slightly lower speedups. This is likely because its much smaller training corpus (119k vs. 252M words; Table 2) limits dictionary coverage. This suggests that fertility is the primary driver of gains, provided the dictionary has adequate coverage.

4.5.4 Dictionary Size

Figure 2 shows mean speedup (averaged across models) versus dictionary size for each minimum probability threshold, with the n-gram configuration fixed at `n123`. Speedup increases quickly with dictionary size up to approximately 100k–200k entries, then flattens, suggesting diminishing returns from adding more entries beyond Lower probability thresholds (especially $p_{\min} = 0.2$) give the highest emulated speedups because they preserve more candidate entries, while stricter thresholds trade coverage for higher-confidence drafts. In terms of memory, even the largest dictionaries (1M entries, trigrams) compress to under 5 MB when stored as a trie. For wall-clock validation we therefore use a compact deployment configuration (200k entries, `n123`, $p_{\min} = 0.8$), which occupies

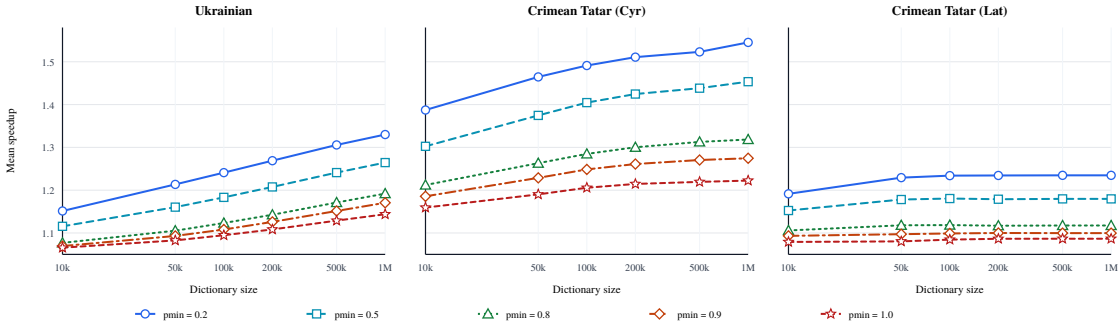

Figure 2: Mean emulated speedup (averaged across models) versus dictionary size for different minimum probability thresholds. N-gram configuration fixed at `n123`.

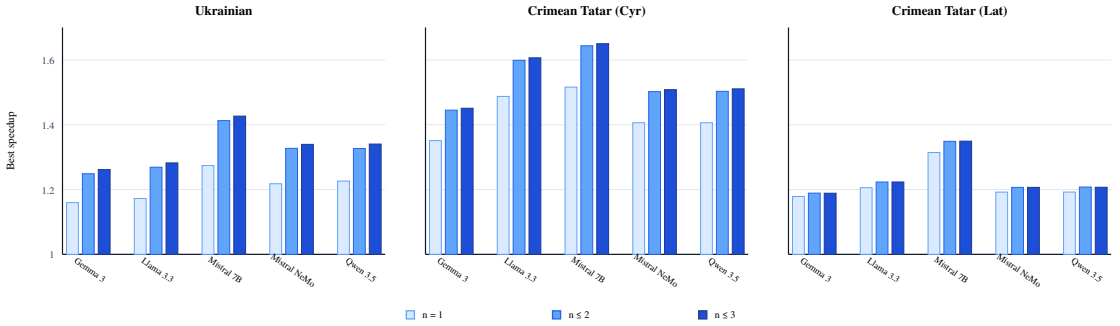

Figure 3: Best emulated speedup by n-gram configuration for each model–language pair.

2–3 MB and keeps the method practical alongside the target model with negligible overhead.

4.5.5 Effect of Longer N-grams

Figure 3 compares `n1`, `n12`, and `n123` for each model–language pair. Adding bigrams (`n12`) consistently improves over unigrams (`n1`), and adding trigrams (`n123`) provides a further but smaller improvement.

4.6 Wall-Clock Validation with vLLM

The emulation study leaves open the question of whether the emulated gains translate to real wall-clock speedups. To answer this, we integrate DictSpec into vLLM (Kwon et al., 2023), an open-source high-throughput serving system, and measure end-to-end generation throughput.

We benchmark five checkpoints listed above on two hardware tiers: Gemma 3 4B, Llama 3.2 3B, and Qwen 3.5 4B on a 16 GB NVIDIA RTX 4090 Laptop GPU; Gemma 3 27B on one 80 GB NVIDIA A100; and Llama 3.3 70B on two 80 GB NVIDIA A100s with tensor parallelism. Models within each family share a tokenizer, so the same trie files from the emulation study apply directly. We use the compact deployment configuration discussed in Section 4.5: `n123`, 200k entries, $p_{\min} = 0.8$. The emulation study uses a speculative budget of $\gamma = 8$ to explore the selected parameter space, but the results show that acceptance drops sharply after the first few draft tokens; we therefore halve the budget to $\gamma = 4$ for the wall-clock experiments.

For each model–language pair we run four configurations: **baseline** (standard autoregressive decoding), **prompt n-gram** speculation using vLLM's built-in prompt-lookup proposer, **DictSpec** (the corpus dictionary alone), and **hybrid** (DictSpec with prompt n-gram fallback). Each configuration serves 20 prompts with a maximum batch size of 5, with a maximum of 2,048 generated tokens per prompt. We report generation throughput (tokens per second) and the token-level acceptance rate for each speculative configuration.

4.6.1 Main Results

Table 5 summarizes the wall-clock benchmarks on coherent outputs. The live results are more

Model	Base tok/s	DictSpec tok/s (×base)	Acc. %	Prompt n-gram tok/s (×base)	Acc. %	Hybrid tok/s (×base)	Acc. %
Ukrainian							
Gemma 3 4B	66.2	66.6 (1.01×)	41.2	75.8 (1.15×)	14.0	**82.8 (1.25×)**	18.0
Llama 3.2 3B	80.7	83.8 (1.04×)	43.6	115.4 (1.43×)	26.6	**122.2 (1.51×)**	28.7
Qwen 3.5 4B	55.6	58.9 (1.06×)	35.2	75.4 (1.36×)	18.4	**75.8 (1.36×)**	18.1
Gemma 3 27B	27.2	27.8 (1.02×)	50.0	29.2 (1.07×)	9.9	**32.1 (1.18×)**	15.9
Llama 3.3 70B	21.1	22.1 (1.05×)	47.8	24.0 (1.14×)	12.9	**28.3 (1.35×)**	23.1
Crimean Tatar (Cyrillic)							
Gemma 3 27B	27.3	29.4 (1.08×)	37.0	**39.5 (1.45×)**	24.0	37.3 (1.37×)	24.7
Llama 3.3 70B	21.2	26.5 (1.25×)	49.0	**45.6 (2.15×)**	46.5	37.4 (1.76×)	36.9
Crimean Tatar (Latin)							
Gemma 3 27B	26.1	26.3 (1.01×)	11.6	**33.9 (1.30×)**	18.3	31.7 (1.22×)	18.5
Llama 3.3 70B	21.2	21.6 (1.02×)	19.4	**29.0 (1.37×)**	24.7	27.4 (1.29×)	18.5

Table 5: Wall-clock generation throughput measured in vLLM on coherent outputs. Throughput columns are reported in tok/s; speculative columns also show speedup over baseline in parentheses. Acc. (%) is the token-level draft acceptance rate. Bold marks the fastest speculative configuration in each row. Models up to 4B run on a 16 GB RTX 4090; Gemma 27B runs on one 80 GB A100; Llama 70B runs on two 80 GB A100s. Dictionary configuration: `n123`, 200k entries, $p_{\min} = 0.8$.

nuanced than the controlled emulation study. Pure DictSpec improves throughput on all coherent rows, but the gains are modest: 1.01–1.25× over baseline. This reflects a practical deployment tradeoff: high acceptance rate is not sufficient by itself; the proposer must also fire often and save enough model steps to overcome lookup and scheduling overhead.

Prompt n-gram speculation is a strong baseline, especially on Crimean Tatar, where the model outputs contain substantial local repetition. The hybrid setting is nevertheless useful: it improves over pure DictSpec on every coherent row and is the fastest configuration for all Ukrainian models. On the larger models, the hybrid reaches 1.18–1.76× speedup, while pure DictSpec reaches 1.01–1.25×.

Manual inspection of the generated outputs reveals that all three smaller models (3–4B parameters) produce severely degenerate text on Crimean Tatar configurations: outputs consist largely of repetitive token loops or wrong-language text (e.g. Kazakh). We therefore omit those rows from Table 5 and use the 3–4B models only for Ukrainian, where their outputs are coherent. The 27B and 70B model outputs are coherent for all three language/script settings.

The wall-clock evaluation refines the interpretation of the method. Pure DictSpec is a high-precision, low-overhead draft source that helps most when tokenizer fertility is high, but as a stand-alone vLLM proposer its live gains are smaller than the emulated reduction in model steps. Prompt n-grams provide strong adaptive coverage when the generated text repeats local context. The best practical deployment is therefore a hybrid view: a small corpus dictionary is a useful complementary draft source that can be combined with prompt-local speculation without changing the target model or its output distribution.

5 Conclusion

We presented DictSpec, a simple method that reduces the inference cost caused by suboptimal tokenization. A lightweight n-gram lookup table, built offline from an unlabeled text corpus, proposes draft token continuations that the target model verifies under the standard speculative decoding protocol, preserving the output distribution exactly. Across seven checkpoints from five tokenizer families and three language/script configurations, emulated DictSpec speedups correlate substantially with tokenizer fertility. In live vLLM serving, pure DictSpec provides modest positive gains on coherent outputs, while a hybrid with prompt-local n-gram speculation reaches up to 1.76×.

The method requires no neural network training, no GPU resources for dictionary construction, and adds less than 5 MB of memory overhead. A single dictionary serves any model that shares the same tokenizer and language/script setting. Thus, DictSpec is most useful as a low-overhead corpus-level draft source for languages whose tokenizers

produce many tokens per word, especially when combined with prompt-local speculation.

Several directions remain for future work. First, combining DictSpec with prompt-based or retrieval-based draft methods (Saxena, 2023; He et al., 2024) could improve coverage and acceptance rates. Second, when the output domain is approximately known (e.g., a medical support system), a domain-specific dictionary could substantially improve acceptance rates. Finally, moving the lookup table to GPU memory, possibly with a custom kernel, could further reduce drafting latency by avoiding CPU-GPU memory communication overhead.

Limitations

DictSpec is most effective when tokenizer fertility is high. For languages with low fertility (e.g., English), the dictionary will match infrequently and gains will be minimal. The method does not improve the quality of the target model's outputs; if the model generates poor-quality text in the target language, our method will not correct this. Similarly, our method does not address the reduced effective context length caused by over-segmentation. Our wall-clock benchmarks cover a limited set of GPU configurations; wall-clock gains may differ under other batched or multi-GPU serving deployments. Finally, this work does not include head-to-head comparisons with approaches that modify the tokenizer itself, such as Lapa LLM (Paniv et al., 2025).

Ethical Considerations

During the preparation of this manuscript, the authors used LLMs extensively to rephrase and polish draft text for improved clarity and readability. The authors reviewed and edited all AI-generated suggestions and take full responsibility for the final content.

Acknowledgments

We thank Dario Stojanovski, Tamara Stankovic and Si-Qing Chen for supporting this work.

References

Orevaoghene Ahia, Sachin Kumar, Hila Gonen, Jungo Kasai, David R. Mortensen, Noah A. Smith, and Yulia Tsvetkov. 2023. Do All Languages Cost the Same? Tokenization in the Era of Commercial Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9904–9923.

Tianle Cai, Yuhong Li, Zhengyang Geng, Hongwu Peng, Jason D. Lee, Deming Chen, and Tri Dao. 2024. Medusa: Simple LLM Inference Acceleration Framework with Multiple Decoding Heads. In *Proceedings of the 41st International Conference on Machine Learning*, pages 5209–5235.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper. 2023. Accelerating Large Language Model Decoding with Speculative Sampling. *arXiv preprint arXiv:2302.01318*.

Jinglin Chen, Qiwei Li, Zuchao Li, Baoyuan Qi, Guoming Liu, Haojun Ai, Hai Zhao, and Ping Wang. 2025. Faster In-Context Learning for LLMs via N-Gram Trie Speculative Decoding. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18040–18051.

Yiming Cui, Ziqing Yang, and Xin Yao. 2023. Efficient and Effective Text Encoding for Chinese LLaMA and Alpaca. *arXiv preprint arXiv:2304.08177*.

Gemma Team. 2025. *Gemma 3 Technical Report*. technical report. Google DeepMind.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, and others. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783*.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason Lee, and Di He. 2024. REST: Retrieval-Based Speculative Decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1582–1595, Mexico City, Mexico.

Jimin Hong, Gibbeum Lee, and Jaewoong Cho. 2024. Accelerating Multilingual Language Model for Excessively Tokenized Languages. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 11095–11111.

Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Tianhao Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. *arXiv preprint arXiv:2310.06825*.

Artur Kiulian, Anton Polishko, Mykola Khandoga, Oryna Chubych, Jack Connor, Raghav Ravishankar, and Adarsh Shirawalmath. 2024. From Bytes to Borsch: Fine-Tuning Gemma and Mistral for the Ukrainian Language Representation. In *Proceedings of the Third Ukrainian Natural Language Processing*

Workshop (UNLP) @ LREC-COLING 2024, pages 83–94, Torino, Italia.

Artur Kiulian, Anton Polishko, Mykola Khandoga, Yevhen Kostiuk, Guillermo Gabrielli, Łukasz Gagała, Fadi Zaraket, Qusai Abu Obaida, Hrishikesh Garud, Wendy Wing Yee Mak, Dmytro Chaplynskyi, Selma Amor, and Grigol Peradze. 2025. From English-Centric to Effective Bilingual: LLMs with Custom Tokenizers for Underrepresented Languages. In *Proceedings of the Fourth Ukrainian Natural Language Processing Workshop (UNLP 2025)*, pages 1–13, Vienna, Austria (online).

Taku Kudo and John Richardson. 2018. SentencePiece: A Simple and Language Independent Subword Tokenizer and Detokenizer for Neural Text Processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast Inference from Transformers via Speculative Decoding. In *Proceedings of the 40th International Conference on Machine Learning*, pages 19274–19286.

Yuhui Li, Fangyun Wei, Chao Zhang, and Hongyang Zhang. 2024. EAGLE: Speculative Sampling Requires Rethinking Feature Uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, pages 28935–28948.

Daniil Maksymenko and Oleksii Turuta. 2025. Tokenization Efficiency of Current Foundational Large Language Models for the Ukrainian Language. *Frontiers in Artificial Intelligence* 8.

Benjamin Minixhofer, Fabian Paischer, and Navid Rekabsaz. 2022. WECHSEL: Effective Initialization of Subword Embeddings for Cross-Lingual Transfer of Monolingual Language Models. In *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 3992–4006.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, John Hoffman, Semarley Jarrett, Kaushik Ram Sadagopan, Dirk Rowe, Shannon Spruit, Chau Tran, Pierre Andrews, Necip Fazil Ayan, Shruti Bhosale, Sergey Edunov, Angela Fan, Cynthia Gao, Vedanuj Goswami, Francisco Guzmán, Philipp Koehn, Alexandre Mourachko, Christophe Ropers, Safiyyah Saleem, Holger Schwenk, and Jeff Wang. 2024. Scaling Neural Machine Translation to 200 Languages. *Nature* 630(8018):841–846.

Thuat Nguyen, Chien Van Nguyen, Viet Dac Lai, Hieu Man, Nghia Trung Ngo, Franck Dernoncourt, Ryan A. Rossi, and Thien Huu Nguyen. 2024. CulturaX: A Cleaned, Enormous, and Multilingual Dataset for Large Language Models in 167 Languages. In *Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024)*, pages 4226–4237.

Yurii Paniv, Bohdan Didenko, Mykola Haltiuk, Vladyslav Humennyy, Andrian Kravchenko, Roman Kyslyi, Viktoriia Makovska, Artem Orlovskyi, Bohdan Ruban, Maksym-Yurii Rudko, Anastasiia Senyk, Nazarii Drushchak, Dmytro Chaplynskyi, and Mariana Romanyshyn. 2025. *Lapa LLM v0.1.2 — the most efficient Ukrainian open-source language model*. version 0.1.2.

Aleksandar Petrov, Emanuele La Malfa, Philip H. S. Torr, and Adel Bibi. 2023. Language Model Tokenizers Introduce Unfairness Between Languages. In *Advances in Neural Information Processing Systems*.

QIRI'M Young. 2024. QIRIM Crimean Tatar Monocorpus. Hugging Face Datasets.

Phillip Rust, Jonas Pfeiffer, Ivan Vulić, Sebastian Ruder, and Iryna Gurevych. 2021. How Good is Your Tokenizer? On the Monolingual Performance of Multilingual Language Models. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3118–3135.

Apoorv Saxena. 2023. Prompt Lookup Decoding.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.

Mitchell Stern, Noam Shazeer, and Jakob Uszkoreit. 2018. Blockwise parallel decoding for deep autoregressive models. *Advances in Neural Information Processing Systems* 31.

Lawrence Stewart, Matthew Trager, Sujan Kumar Gonugondla, and Stefano Soatto. 2024. The N-Grammys: Accelerating Autoregressive Inference with Learning-Free Batched Speculation. *arXiv preprint arXiv:2411.03786*.

Qwen Team. 2026. Qwen3.5: Accelerating Productivity with Native Multimodal Agents.

Susumu Yata. 2023. *MARISA: Matching Algorithm with Recursively Implemented StorAge*.

A Speculative Decoding Visualizations

The following figures visualize speculative decoding outcomes. Each token is color-coded: Accepted draft tokens were proposed by the dictionary and accepted by the target model; Rejected (target corrected) tokens were proposed but replaced by the target model; Bonus (target +1) tokens are additional tokens generated by the target model after accepting a draft; and No draft available tokens had no dictionary match and were decoded autoregressively.

A.1 Ukrainian

Figure 4: Speculative decoding for Ukrainian using Gemma 3 27B. Draft acceptance rate is 42% for this excerpt; speedup is 1.22×

A.2 Crimean Tatar (Cyrillic)

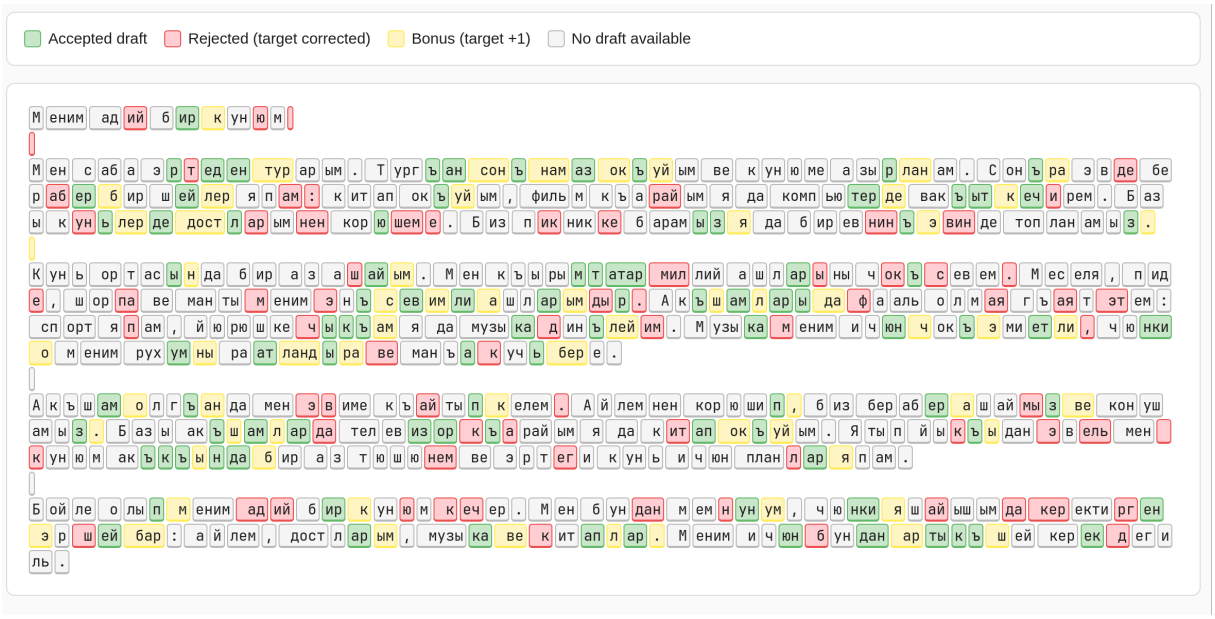

Figure 5: Speculative decoding for Crimean Tatar (Cyrillic script) using Llama 3.3 70B. Draft acceptance rate is 33%; speedup is 1.20×

A.3 Crimean Tatar (Latin)

Figure 6: Speculative decoding for Crimean Tatar (Latin script) using Llama 3.3 70B. Draft acceptance rate is 16%; speedup is 1.09×

B Evaluation Prompts

The following are examples of prompts used in the wall-clock validation experiments (Section 4.6). For each model–language pair, these prompts are sent to the vLLM serving engine as chat-formatted requests, and the model generates up to 2,048 tokens per prompt. The prompts are designed to elicit long-form text in the target language and script.

B.1 Ukrainian

1. Напиши історію України за період 1991–2026 років у десяти параграфах.
2. Напиши короткий текст (3–4 параграфи) про свій звичайний день, використовуючи всі сім відмінків української мови. Після кожного абзацу вкажи, яке слово в якому відмінку вжито.
3. Напиши есе на 1000 слів про значення творчості Лесі Українки для сучасної української ідентичності. Наведи конкретні приклади з її творів.
4. Напиши два варіанти одного й того самого повідомлення: (а) офіційний лист до міського голови Петренко Віктора Івановича з проханням відремонтувати дорогу, і (б) повідомлення другу в месенджері про ту саму проблему. Використовуй відповідний стиль, звертання та лексику для кожного варіанту.

B.2 Crimean Tatar (Cyrillic)

1. Къырымтатар халкъынынъ тарихыны он параграфта язынъыз. Алтын Орда девиринден бугуньге къадар олгъан энъ муим вакъиаларны анълатынъыз.
2. Исмаил Гаспринскийнинъ къырымтатар миллетининъ инкишафындаки ролю акъкъында бинъ сёзлюк бир эссе язынъыз. Онынъ «Терджиман» газетасындан ве дигер фаалиетлеринден мисаллер кетиринъиз.
3. Къырымтатар тилинде сизинъ адий бир кунюнъизни тасвирлеген 3–4 параграфлыкъ бир язы язынъыз.
4. Эки вариантта бир хабер язынъыз: (а) мектеп мудирине расмий бир мектюп ве (б) аркъадашынъызгъа месажда айны меселе акъкъында язув. Эр вариант ичюн мунасип услюп ве сёзлер къулланынъыз.

B.3 Crimean Tatar (Latin)

1. Qırımtatar halqınıñ tarihını on paragrafta yazıñız. Altın Orda devirinden bugünge qadar olğan eñ muim vaqialarnı añlatıñız.
2. İsmail Gasprinskiyniñ qırımtatar milletiniñ inkişafındaki rolü aqqında biñ sözlük bir esse yazıñız. Onıñ «Terciman» gazetasından ve diger faaliyetlerinden misaller ketiriñiz.
3. Qırımtatar tilinde siziñ adiy bir künüñizni tasvirlegen 3–4 paragraflıq bir yazı yazıñız.
4. Eki variantta bir haber yazıñız: (a) mektep müdirine resmiy bir mektüp ve (b) arqadaşıñızğa mesajda aynı mesele aqqında yazuv. Er variant içün münasip üslüp ve sözler qullanıñız.