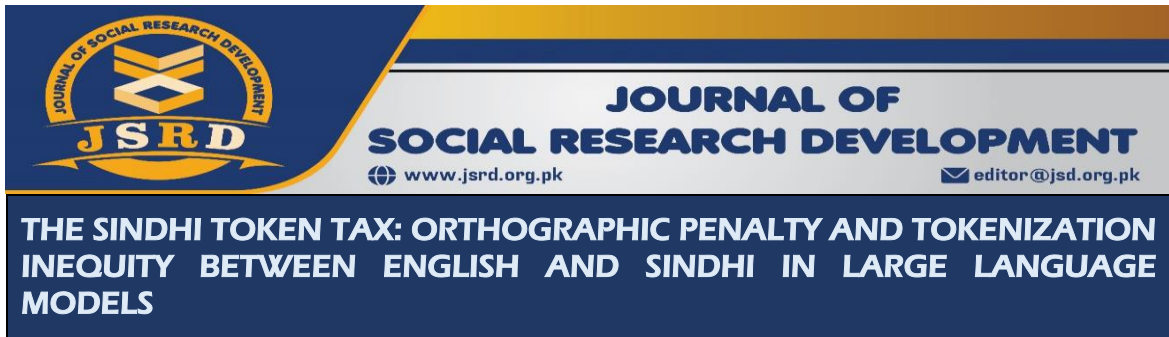




Dr. Ali Siddiqui¹, Dr. Shauban Ali Solangi², Uzma Ehsan³

¹English, Hyderabad Institute for Technology and Management Sciences (HITMS), Hyderabad, Sindh, Pakistan.

Email: drali@faculty.hitms.edu.pk

²Department of Computer Science & Related Studies, Hyderabad Institute for Technology and Management Sciences (HITMS), Hyderabad, Sindh, Pakistan. Email: dr.shauban@faculty.hitms.edu.pk

³Senior Lecturer, NUST-Military College of Signals. Email: uzmaehsan@mcs.nust.edu.pk

KEYWORDS	ABSTRACT
computational linguistics; tokenization; large language models; Sindhi; English; token premium; Orthographic Penalty Index; language equity; Pakistan	Large language models (LLMs) do not read words; they read tokens, and the tokenizer decides how many tokens a language needs to say the same thing. This paper introduces the notion of a Sindhi token tax and tests it empirically for Pakistan's English-Sindhi context. Using 1,004 parallel sentences from the FLORES-200 benchmark in English, Sindhi and Urdu, the study measured five widely deployed tokenizers (GPT-2, Llama 2, Llama 3, Qwen2 and Command-R) with a reproducible Python pipeline. A convergent mixed-methods design combined quantitative indices with a rule-assisted qualitative analysis of how frequent Sindhi words are segmented. Quantitatively, Sindhi required 2.87 to 4.98 times as many tokens as English for the same content in total (Wilcoxon signed-rank, $r = .87$, $p < .001$), and a 4,096-token context window held about 770 to 1,335 Sindhi words against 2,890 to 3,330 English words. A new Orthographic Penalty Index showed that the 18 letters unique to the Sindhi alphabet, such as ڪ, ڙ, ڻ and ڦ, were split into byte fragments in 100% of their occurrences in four of the five tokenizers, while letters shared with Arabic and Urdu were almost never split. Qualitatively, five patterns emerged, including the fragmentation of core grammatical words, the "Urdu shortcut" by which Urdu code points are cheaper than Sindhi ones, and the invisibility of Sindhi morphology to subword boundaries. The study also found Unicode inconsistency in digital Sindhi, with 238 of 1,004 benchmark sentences mixing forms of the letter heh. The paper proposes the Script-Aware Tokenization Equity Framework (SATEF) and argues that script-level design, not only data scarcity, makes AI costlier and weaker for Sindhi speakers, with implications for language policy, education and technology in Pakistan.
ARTICLE HISTORY	 2026 Journal of Social Research Development
Date of Submission: 18-08-2026	
Date of Acceptance: 21-09-2026	
Date of Publication: 01-10-2026	
Correspondence	
Email:	drali@faculty.hitms.edu.pk
DOI	https://doi.org/10.59075/jsrd.v7i8.547
Volume/Issue	7(8), 2026, 01-26

Introduction

Background

Computational linguistics in the 2020s is dominated by large language models. Systems built on transformer architectures now translate, summarise, tutor and converse in dozens of languages, and they are rapidly entering classrooms, offices and public services across South Asia (Dubey et al., 2024; Touvron et al., 2023; Yang et al., 2024). Behind every such system sits a component that users rarely see: the *tokenizer*. Before a model can process a sentence, the tokenizer cuts it into units called tokens, drawn from a fixed vocabulary learned mainly from the data on which the model was trained (Kudo & Richardson, 2018; Sennrich et al., 2016). The number of tokens a sentence needs determines how much it costs to process, how much of it fits into the model's context window, and, indirectly, how well the model learns the language (Mielke et al., 2021; Rust et al., 2021).

Because tokenizer vocabularies are learned from data dominated by English and a small number of other high-resource languages, they encode English efficiently and many other languages inefficiently. Petrov et al. (2023) showed that the same text can require many times more tokens in some languages than in English, and Ahia et al. (2023) demonstrated that speakers of such languages pay more for commercial language-model services while receiving poorer results. Global surveys of language technology confirm that the benefits of NLP are distributed very unevenly across the world's languages (Blasi et al., 2022; Joshi et al., 2020; Ranathunga & de Silva, 2022).

The Pakistani context: English and Sindhi

Pakistan is a multilingual country in which English holds the highest institutional prestige, Urdu serves as the national language, and regional languages such as Sindhi, Punjabi, Pashto and Balochi carry the everyday lives of millions (Mansoor, 2005; Rahman, 1996, 2002). Sindhi is the language of Sindh province, has official status there, and possesses a long literary and journalistic tradition; it is spoken by tens of millions of people (Pakistan Bureau of Statistics, 2023; Rahman, 1996). In computational terms, however, Sindhi is a low-resource language. Its digital corpora, annotated datasets and language tools remain limited compared with English and even Urdu (Ali et al., 2019; Jamro, 2017; Motlani, 2016).

Sindhi is written in a Perso-Arabic script that extends the Arabic alphabet with letters for sounds that Arabic, Persian and Urdu lack: implosives such as ٻ, ڄ, ڳ and ڙ, retroflex consonants such as ٺ, ڍ and ڏ, aspirated forms such as ٽ and ٿ, and distinctive letters such as ڪ (Unicode Consortium, 2023). These letters are central to Sindhi grammar and vocabulary; the everyday verb ڪرڻ (to do) contains two of them. If a tokenizer's vocabulary contains the Arabic and Urdu letters but not the Sindhi ones, Sindhi text will be penalised at the most basic level of the writing system. This paper calls that penalty the *orthographic penalty*, and the extra cost it produces the *Sindhi token tax*.

Pakistan's own policy agenda gives the question urgency. The Digital Pakistan Policy commits the state to digital inclusion and e-services for all citizens (Ministry of Information Technology and Telecommunication, 2018). If AI services cost more and work worse in Sindhi than in English, digital inclusion will reproduce the old linguistic hierarchy in a new technological form (Bender et al., 2021; Rahman, 2002).

The problem

Existing studies of tokenization inequity treat languages in large multilingual samples and rarely examine the internal structure of particular scripts (Ahia et al., 2023; Petrov et al., 2023). Research on Sindhi computational linguistics has concentrated on corpora, part-of-speech tagging, word embeddings and spell checking (Ali et al., 2019; Jamro, 2017; Motlani, 2016) rather than on how contemporary LLM tokenizers treat the language. No study, to the author's knowledge, has measured the tokenization of Sindhi against English and Urdu across several modern LLM tokenizers, or asked which features of the Sindhi script drive the cost.

The new idea

This paper advances three original contributions. First, it proposes the concept of the **Sindhi token tax**: the extra computational and economic cost that a Sindhi speaker incurs, relative to an English speaker, for communicating the same content with an LLM. Second, it introduces the **Orthographic Penalty Index (OPI)**, a script-level measure that compares the fragmentation of letters unique to a language's alphabet with the fragmentation of letters it shares with better-resourced languages. Third, it proposes the **Script-Aware Tokenization Equity Framework (SATEF)**, which links orthographic, segmentation, economic and representational equity in a single cascade.

Research questions

1. **RQ1.** How large is the token premium for Sindhi relative to English, and to Urdu, across contemporary LLM tokenizers?
2. **RQ2.** To what extent do letters unique to the Sindhi alphabet drive fragmentation, as measured by the Orthographic Penalty Index?
3. **RQ3.** How do tokenizers segment frequent Sindhi words, and what linguistic patterns does this segmentation reveal?
4. **RQ4.** What are the implications for AI access, language policy and education in Pakistan?

Significance

The study contributes to computational linguistics by adding a script-level dimension to the measurement of tokenization inequity, and to English linguistics in Pakistan by showing how the dominance of English is being reproduced inside the architecture of AI systems. Because the

analysis uses an open benchmark and public tokenizer vocabularies, every figure in this paper can be reproduced by other researchers (Goyal et al., 2022; NLLB Team et al., 2022).

Review of Literature

From words to subwords

Early statistical NLP operated on whole words, which forced systems to treat any unseen word as unknown. Sennrich et al. (2016) adapted byte-pair encoding (BPE) to neural machine translation: starting from characters, the algorithm repeatedly merges the most frequent adjacent pair of symbols until it reaches a target vocabulary size. Kudo (2018) proposed the unigram language-model tokenizer, and Kudo and Richardson (2018) released SentencePiece, which treats text as a raw stream of Unicode characters and has become the standard toolkit for multilingual models. GPT-2 introduced *byte-level* BPE, in which the base alphabet is the 256 possible bytes of UTF-8, so that any string can be encoded without unknown tokens (Radford et al., 2019). Mielke et al. (2021) review this history and note that subword tokenization is a compromise: it guarantees coverage, but the resulting segments need not correspond to meaningful linguistic units.

Tokenizer quality and model performance

Tokenization is not a neutral pre-processing step. Rust et al. (2021) showed that multilingual models perform worse on languages whose tokenizers produce high *fertility*, the average number of tokens per word, and that replacing a multilingual tokenizer with a dedicated monolingual one improves downstream results. Limisiewicz et al. (2023) found that the way vocabulary is allocated across languages affects multilingual performance, and Liang et al. (2023) showed that simply enlarging the vocabulary to one million tokens (XLM-V) benefits low-resource languages. At the other extreme, Xue et al. (2022) proposed ByT5, a token-free model operating directly on bytes, which is robust to spelling variation but computationally expensive because sequences become long.

Tokenization as an equity problem

A recent line of research reframes tokenization as a matter of fairness. Petrov et al. (2023) compared 17 tokenizers on the parallel FLORES-200 corpus and found that some languages need more than ten times as many tokens as English for the same text, which they call a *premium*; they recommend that tokenizers be trained to achieve *parity* across languages. Ahia et al. (2023) showed that commercial APIs charge by the token, so speakers of high-premium languages pay more for the same service and fit less text into the model's context. These findings connect to broader critiques of language technology: Joshi et al. (2020) classified most of the world's languages as "left behind" by NLP resources, Blasi et al. (2022) documented systematic performance gaps across languages that correlate with economic and demographic power, and Bender et al. (2021) warned that the costs and risks of large models fall unevenly on marginalised communities.

Scripts, Unicode and byte fragmentation

Most equity studies measure tokens per sentence without asking *why* some languages fragment more. One reason lies in the script. In UTF-8, Latin letters take one byte, whereas Arabic-script letters take two bytes (Unicode Consortium, 2023). A byte-level tokenizer that has not learned a merged token for a particular Arabic-script letter must represent it as two separate byte tokens, neither of which is a meaningful character (Radford et al., 2019; Xue et al., 2022). SentencePiece tokenizers face a parallel problem through *byte fallback*: characters outside the learned vocabulary are encoded as raw bytes (Kudo & Richardson, 2018). The Arabic block of Unicode contains many letters used only by particular languages, including the Sindhi letters in the range U+0680 to U+06BF, and these are likely to be rare in training data dominated by Arabic, Persian and Urdu. The script, therefore, can impose a penalty on top of the general scarcity of data.

Computational linguistics for Sindhi and Urdu in Pakistan

Urdu language processing in Pakistan has developed through sustained institutional work on corpora, lexicons and tools (Hussain, 2008). Sindhi has received less attention. Jamro (2017) surveyed Sindhi language processing and identified the scarcity of annotated corpora, script complexity and the coexistence of Perso-Arabic and Devanagari writing as major obstacles. Motlani (2016) described the development of basic tools such as tokenisers and morphological resources for Sindhi and characterised it as a resource-poor language. Ali et al. (2019) built a large Sindhi corpus from news and web sources and trained word embeddings, noting the need to normalise inconsistent spellings and character encodings. Multilingual benchmarks have since begun to include Sindhi: FLORES-200 provides professionally translated sentences in more than 200 languages, including Sindhi in Arabic script (Goyal et al., 2022; NLLB Team et al., 2022), and SIB-200 adds topic labels to the same sentences (Adelani et al., 2024). Language policy research, however, reminds us that the digital fate of Sindhi is shaped by the same hierarchy that places English above Urdu and Urdu above regional languages in Pakistani institutions (Mansoor, 2005; Rahman, 1996, 2002).

Research gap

Three gaps follow from this review. First, tokenization-equity studies have not examined Sindhi specifically or compared it with both English and Urdu within the Pakistani context. Second, no study has isolated the contribution of language-specific letters to tokenization cost. Third, there is no framework that connects script-level encoding with segmentation, cost and representation in a way that can guide policy for a multilingual country such as Pakistan.

Theoretical framework: the Script-Aware Tokenization Equity Framework (SATEF)

This paper proposes the Script-Aware Tokenization Equity Framework (SATEF) as an original synthesis of the literature reviewed above. It organises tokenization equity into four layers, arranged as a cascade from the writing system upward (Figure 1).

Figure 1. Script-Aware Tokenization Equity Framework (SATEF)

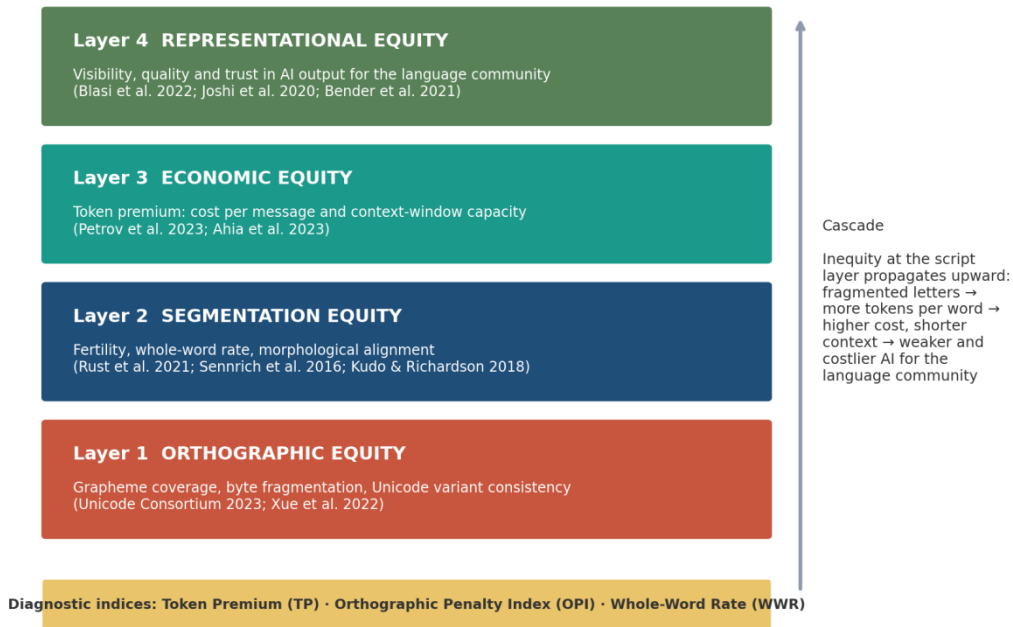


Figure 1. The Script-Aware Tokenization Equity Framework (SATEF)

Layer 1, Orthographic equity, asks whether the letters of a language’s alphabet are represented as whole units in the tokenizer’s vocabulary, whether they are split into bytes, and whether the language’s digital text uses Unicode consistently (Unicode Consortium, 2023; Xue et al., 2022). It is measured here by the Orthographic Penalty Index.

Layer 2, Segmentation equity, asks how many tokens a word needs, how often frequent words are kept whole, and whether token boundaries respect morphological structure (Kudo & Richardson, 2018; Rust et al., 2021; Sennrich et al., 2016).

Layer 3, Economic equity, asks how much more a speaker pays per message and how much less text fits into a model’s context window (Ahia et al., 2023; Petrov et al., 2023).

Layer 4, Representational equity, asks whether the resulting models serve the language community with comparable quality, visibility and trust (Bender et al., 2021; Blasi et al., 2022; Joshi et al., 2020).

The arrow in Figure 1 expresses the framework’s central claim: inequity introduced at the orthographic layer propagates upward. Fragmented letters produce more tokens per word; more tokens per word mean higher costs and shorter effective context; and higher costs and poorer segmentation contribute to weaker, more expensive AI for the language community. SATEF adds

three diagnostic indices: the Token Premium (TP), the Orthographic Penalty Index (OPI) and the Whole-Word Rate (WWR). The present study measures Layers 1 to 3 directly and discusses Layer 4 on the basis of prior research.

Research Methodology

Research design

The study follows a convergent mixed-methods design in which a quantitative computational experiment and a qualitative analysis of segmentation are conducted separately and integrated at the interpretation stage (Creswell & Plano Clark, 2018). The quantitative strand measures token counts, fertility, fragmentation and the OPI across languages and tokenizers. The qualitative strand examines how individual Sindhi words are segmented and interprets the patterns linguistically, using a codebook and rule-assisted coding followed by close reading (Braun & Clarke, 2006).

Data

The study uses the FLORES-200 evaluation benchmark in the version distributed with the SIB-200 topic-classification dataset (Adelani et al., 2024; NLLB Team et al., 2022). FLORES sentences were drawn from English Wikimedia sources and professionally translated into each language, so the same meaning is expressed in every language (Goyal et al., 2022). Three languages were selected: English (eng_Latn), Sindhi in Arabic script (snd_Arab) and Urdu (urd_Arab), the latter serving as a Pakistani comparison language that shares most of the Perso-Arabic alphabet with Sindhi. The training, development and test splits were combined and aligned by sentence identifier, giving 1,004 parallel sentence triples (Table 1).

Table 1. Corpus description (FLORES-200 via SIB-200)

Language	Script	Sentences	Words	Characters
English	Latin	1,004	21,551	129,692
Sindhi	Perso-Arabic (Sindhi)	1,004	24,767	117,817
Urdu	Perso-Arabic (Urdu)	1,004	27,447	129,181

Note. Words are whitespace-separated units. Topic labels: science/technology (252), travel (198), politics (146), sports (122), health (110), entertainment (93), geography (83).

Tokenizers

Five tokenizers were selected to represent the main families and generations of LLM tokenization (Table 2). Their vocabularies were obtained from public sources: the GGUF vocabulary files distributed with the open-source llama.cpp project (ggml-org, 2024) and the SentencePiece model released with the llama2.c project (Karpathy, 2023). Byte-level BPE tokenizers were reconstructed with the Hugging Face *tokenizers* library (Wolf et al., 2020) from their published vocabularies, merge lists and pre-tokenization rules; the Llama 2 tokenizer was run directly with

SentencePiece (Kudo & Richardson, 2018). As a validity check, the control phrase *Hello world* was encoded with the reference token identifiers of the original tokenizers (for example, 15496 and 995 for GPT-2, and 9906 and 1917 for Llama 3). For Command-R, whose pre-tokenization rules are not fully documented, the GPT-2 splitting rule was used; this rule affects digits and punctuation rather than Arabic-script letters, so its effect on Sindhi counts is negligible.

Table 2. Tokenizers analysed

Tokenizer	Family	Vocabulary size	Model reference
GPT-2 (2019)	Byte-level BPE	50,257	Radford et al. (2019)
Llama 2 (2023)	SentencePiece BPE with byte fallback	32,000	Touvron et al. (2023)
Llama 3 (2024)	Byte-level BPE	128,256	Dubey et al. (2024)
Qwen2 (2024)	Byte-level BPE	151,936	Yang et al. (2024)
Command-R (2024)	Byte-level BPE	256,000	Aryabumi et al. (2024)

Quantitative measures

Six measures were computed for every sentence, language and tokenizer. **Tokens per sentence** is the raw count. **Token premium (TP)** is the number of tokens for a Sindhi (or Urdu) sentence divided by the number for its English equivalent (Petrov et al., 2023). **Fertility** is the number of tokens per whitespace-separated word (Rust et al., 2021). **Effective context capacity** is the number of words that fit into a 4,096-token window, calculated as 4,096 divided by fertility (Ahia et al., 2023). **Fragmentation rate** is the percentage of non-space characters whose UTF-8 bytes are divided between two or more tokens. The **Orthographic Penalty Index (OPI)** is the fragmentation rate of the 18 letters unique to the Sindhi alphabet (پ ٻ ٺ ڻ ڄ ڇ ڙ ڍ ڏ ڌ ڇ ڦ ڪ ڳ ڳ ڻ) minus the fragmentation rate of 32 letters Sindhi shares with the Arabic, Persian and Urdu alphabets, measured over their occurrences in running Sindhi text:

$$OPI = F_{\text{Sindhi-specific}} - F_{\text{shared}}$$

An OPI of 0 means Sindhi's own letters are treated like shared letters; an OPI of 100 means every occurrence of a Sindhi-specific letter is fragmented while no shared letter is. Letter-level encoding was also tested in isolation. Finally, the Sindhi corpus was scanned for alternative Unicode code points for the letters heh and yeh.

Statistical analysis

Because token premiums are ratios with skewed distributions, non-parametric tests were used. Wilcoxon signed-rank tests compared paired Sindhi and English, and Sindhi and Urdu, token counts for each tokenizer, with effect size $r = z/\sqrt{N}$ (Wilcoxon, 1945; Cohen, 1988). A Friedman test

compared the Sindhi premium across the five tokenizers, with Kendall's W as effect size (Friedman, 1937). A Kruskal–Wallis test compared the premium across the seven topic domains, and chi-square tests compared the fragmentation of Sindhi and Urdu characters.

Qualitative strand

The 80 most frequent Sindhi word types in the corpus (excluding numerals and Latin-script tokens) were segmented by each tokenizer, giving 400 segmentations. Each was coded with a five-category codebook (Table 5): *whole word*; *morph-aligned*, where a word is split into full characters and the final piece matches a Sindhi inflectional ending such as *-ڻ*, *-ڻ*, *-ڻ*, *-ڻ* or *-ڻ*; *arbitrary sub-word*; *character-level*; and *byte-fragmented*, where at least one piece is a partial character. A first pass was applied automatically by rules, after which the segmentations were read closely and interpreted, following the familiarisation, coding and theme-generation phases of thematic analysis (Braun & Clarke, 2006). One representative sentence was also visualised across all tokenizers.

Software and reproducibility

The pipeline was written in Python 3, using pandas for data management (McKinney, 2010), SciPy for statistics (Virtanen et al., 2020), Matplotlib for charts (Hunter, 2007), the *tokenizers* library (Wolf et al., 2020), SentencePiece (Kudo & Richardson, 2018), and the Pillow imaging library with the *libraqm* text-layout engine to render correctly shaped Sindhi script. All data and vocabularies are public, and the scripts accompany this paper, so every result can be regenerated.

Ethical considerations

The study analyses published benchmark sentences and public software artefacts; it involves no human participants and no personal data. The FLORES-200 and SIB-200 datasets are released under open licences for research use (Adelani et al., 2024; NLLB Team et al., 2022).

Data Collection and Analysis

All results in this section were computed with the Python pipeline described in Section 3.7. Part A reports the quantitative strand (Figures 2 to 9); Part B reports the qualitative strand (Figures 10 and 11). Each figure is followed by a quantitative interpretation, which states what the numbers show, and a qualitative reading, which explains what they mean linguistically.

Part A: Quantitative Analysis

How many tokens does the same content need? (RQ1)

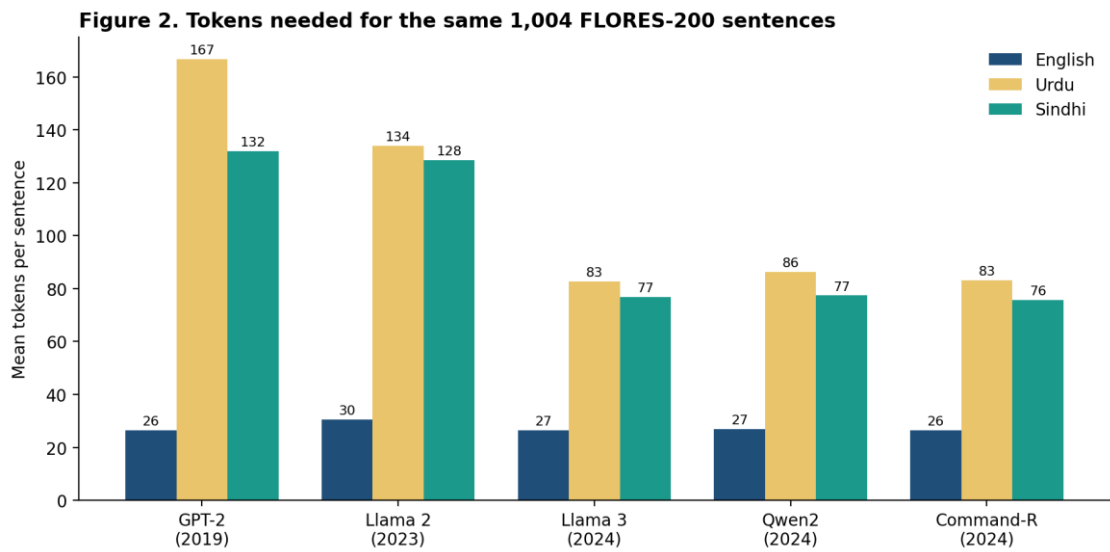


Figure 2. Mean tokens per sentence for English, Urdu and Sindhi across five tokenizers

Quantitative interpretation of Figure 2. For English, all five tokenizers needed between 26 and 30 tokens per sentence on average. For Sindhi, GPT-2 needed 132 tokens and Llama 2 needed 128, while the three 2024 tokenizers needed 76 to 77. Urdu followed a similar pattern at a slightly higher level (83 to 167 tokens). In total, the 1,004 English sentences required 26,513 to 30,563 tokens, whereas the same sentences in Sindhi required 75,990 to 132,374 tokens (Table 3).

Qualitative reading of Figure 2. English is encoded almost identically by every tokenizer, from 2019 to 2024, which shows that English efficiency was already solved in the earliest model. Sindhi improved substantially between the older and newer tokenizers, yet even the best 2024 tokenizers need about three tokens for every English one. Progress for Sindhi has been real but incomplete, and it has come as a side effect of larger multilingual vocabularies rather than deliberate design for the language (Dubey et al., 2024; Yang et al., 2024).

Table 3. Core tokenization metrics by tokenizer and language (1,004 sentences)

Tokenizer	Language	Total tokens	Fertility	Characters per token	Fragmented characters (%)	Words per 4,096 tokens
GPT-2	English	26,598	1.23	4.88	0.03	3,319
GPT-2	Sindhi	132,374	5.35	0.89	37.2	766

GPT-2	Urdu	167,427	6.10	0.77	56.4	671
Llama 2	English	30,563	1.42	4.24	0.0	2,888
Llama 2	Sindhi	128,952	5.21	0.91	12.1	787
Llama 2	Urdu	134,408	4.90	0.96	5.4	836
Llama 3	English	26,620	1.24	4.87	0.0	3,316
Llama 3	Sindhi	77,058	3.11	1.53	12.6	1,316
Llama 3	Urdu	83,087	3.03	1.56	14.4	1,353
Qwen2	English	27,085	1.26	4.79	0.0	3,259
Qwen2	Sindhi	77,690	3.14	1.52	3.9	1,306
Qwen2	Urdu	86,635	3.16	1.49	0.1	1,298
Command-R	English	26,513	1.23	4.89	0.0	3,329
Command-R	Sindhi	75,990	3.07	1.55	12.6	1,335
Command-R	Urdu	83,467	3.04	1.55	16.5	1,347

The token premium

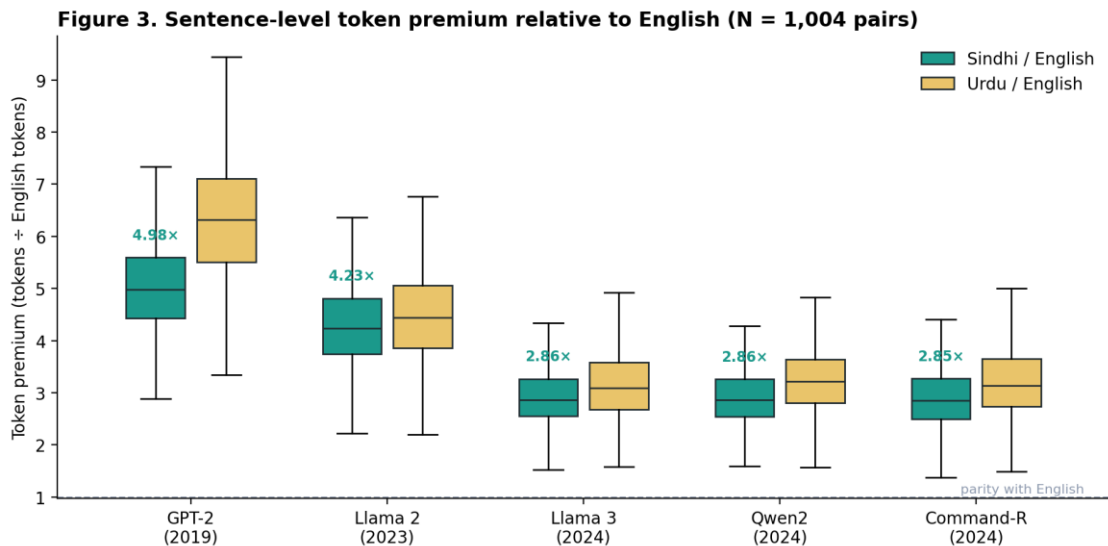


Figure 3. Distribution of the sentence-level token premium for Sindhi and Urdu relative to English

Quantitative interpretation of Figure 3. The median Sindhi premium was 4.98 for GPT-2, 4.23 for Llama 2, 2.86 for Llama 3, 2.86 for Qwen2 and 2.85 for Command-R. No single sentence in any tokenizer reached parity with English; even the lower whiskers stay well above the dashed parity

line. Wilcoxon signed-rank tests confirmed that Sindhi needed more tokens than English in every tokenizer ($z = 27.45$, $p < .001$, $r = .87$; Table 4); in fact every one of the 1,004 Sindhi sentences needed more tokens than its English equivalent. A Friedman test showed that the premium differed strongly across tokenizers, $\chi^2(4) = 3,238.19$, $p < .001$, Kendall's $W = .81$. Urdu's median premium was slightly higher than Sindhi's (3.09 to 6.32), and Sindhi sentences needed fewer tokens than Urdu ones in all five tokenizers (median differences of 5 to 32 tokens).

Table 4. Wilcoxon signed-rank tests on paired sentence token counts (N = 1,004)

Tokenizer	Sindhi vs English: median difference	z	r	Sindhi vs Urdu: median difference	z	r
GPT-2	+101	27.45	.87	-32	26.71	.84
Llama 2	+94	27.45	.87	-5	9.48	.30
Llama 3	+48	27.45	.87	-6	12.34	.39
Qwen2	+48	27.45	.87	-8	19.61	.62
Command-R	+47	27.45	.87	-7	14.69	.46

Note. All $p < .001$. For Sindhi vs English, every sentence pair showed a positive difference, so z reaches the same ceiling value in all tokenizers.

Qualitative reading of Figure 3. The premium is best understood as a price. If a service charges by the token, a Sindhi speaker pays roughly three times as much as an English speaker for the same message with current tokenizers, and five times as much with GPT-2 (Ahia et al., 2023; Petrov et al., 2023). This is the *Sindhi token tax*. The comparison with Urdu is revealing: Sindhi is not worse than Urdu at the sentence level, partly because the Urdu translations are longer (27,447 words against 24,767). The disadvantage both languages share is relative to English, which reflects the English-centred composition of training data and the prestige hierarchy of Pakistani languages (Joshi et al., 2020; Rahman, 2002).

Fertility and effective context

Figure 4. Fertility and effective context capacity by language

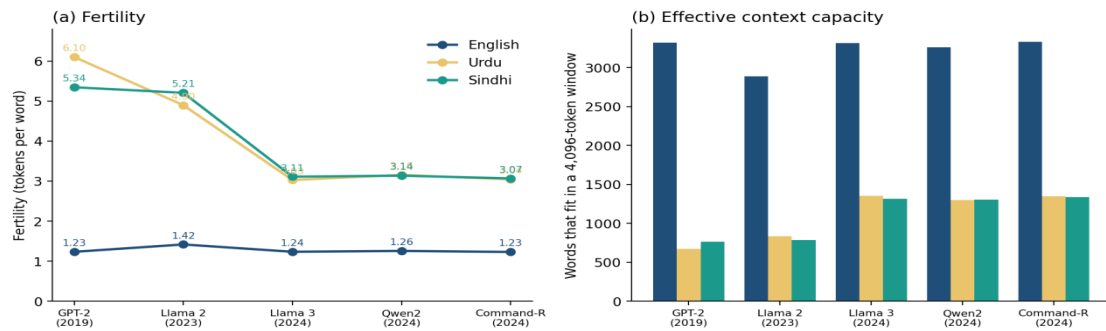


Figure 4. (a) Fertility and (b) effective context capacity by language and tokenizer

Quantitative interpretation of Figure 4. English fertility was almost constant at 1.23 to 1.42 tokens per word. Sindhi fertility fell from 5.35 (GPT-2) and 5.21 (Llama 2) to 3.07 to 3.14 in the 2024 tokenizers, closely tracking Urdu. In a 4,096-token context window, a model can therefore hold about 3,300 English words but only 766 to 1,335 Sindhi words. Put differently, a Sindhi reader receives between 23% and 40% of the context capacity an English reader receives.

Qualitative reading of Figure 4. Context capacity determines what an AI system can “see” at once: a long legal document, a school textbook chapter or a conversation history. A Sindhi user who wants a model to summarise a document, answer questions about a policy text or keep track of a long tutoring dialogue runs out of context roughly three times sooner than an English user. For education in Sindh, where textbooks and assessments are produced in Sindhi, this is a practical limitation, not merely a technical statistic (Rust et al., 2021).

Character fragmentation

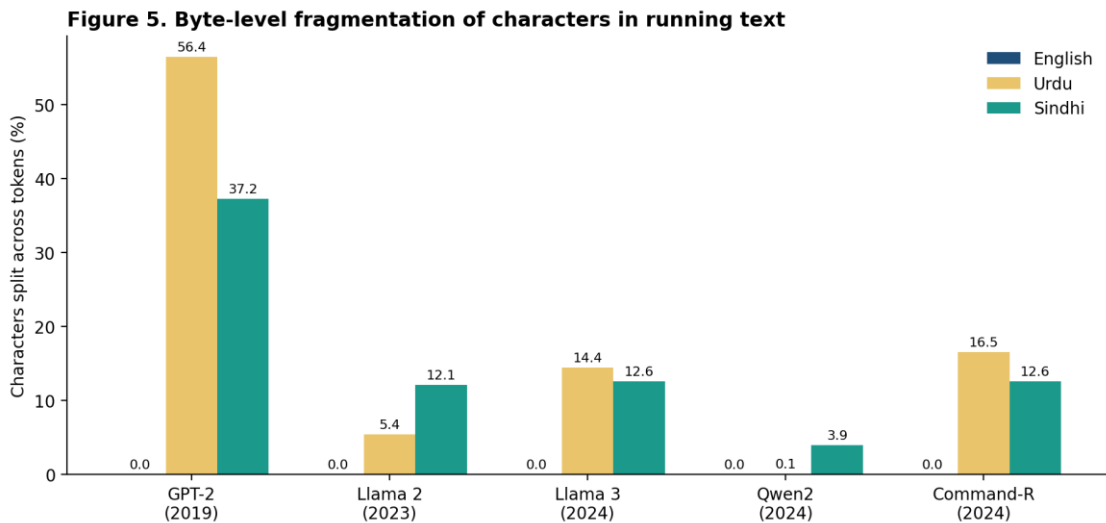


Figure 5. Percentage of characters split across tokens in running text

Quantitative interpretation of Figure 5. English characters were essentially never split (at most 0.03%). For Sindhi, 37.2% of characters were split into byte fragments by GPT-2, around 12% by Llama 2, Llama 3 and Command-R, and 3.9% by Qwen2. Urdu fragmentation followed a different profile: very high in GPT-2 (56.4%), moderate in Llama 3 (14.4%) and Command-R (16.5%), and almost zero in Qwen2 (0.1%). Chi-square tests showed that Sindhi and Urdu fragmentation differed significantly in every tokenizer (χ^2 from 145.9 to 7,277.2, all $p < .001$). Sindhi fragmented more than Urdu in Llama 2 and Qwen2, and less in the others.

Qualitative reading of Figure 5. The Qwen2 result is the key to interpreting this figure. Qwen2 represents almost every Urdu character whole, yet still fragments 3.9% of Sindhi characters. The

residual fragmentation must therefore come from something Sindhi has and Urdu does not, which points directly to the letters unique to the Sindhi alphabet. The next two figures test this explanation.

Sindhi letters in isolation (RQ2)

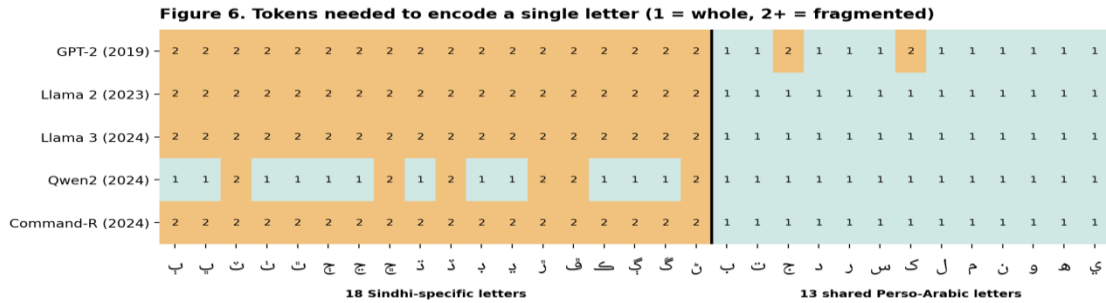


Figure 6. Tokens needed to encode each letter in isolation

Quantitative interpretation of Figure 6. Each cell shows how many tokens a single letter needs. In GPT-2, Llama 2, Llama 3 and Command-R, every one of the 18 Sindhi-specific letters needed two tokens, meaning none exists as a whole token in these vocabularies (0% single-token coverage). Shared letters were whole in almost every case (96.9% to 100% in the newer tokenizers; 40.6% in GPT-2). Qwen2 was the exception: it encoded 12 of the 18 Sindhi-specific letters (66.7%) as single tokens, but still fragmented six, including ٺ, ٽ and ڙ.

Qualitative reading of Figure 6. The dividing line in the heat map is not a line between rich and poor languages; it is a line inside a single script. The letters that Sindhi shares with Arabic, Persian and Urdu are protected by those languages' presence in training data. The letters that belong to Sindhi alone are left out. The fact that Qwen2 includes most of them shows that this is a design outcome, not a technical necessity (Kudo & Richardson, 2018; Liang et al., 2023).

The Orthographic Penalty Index

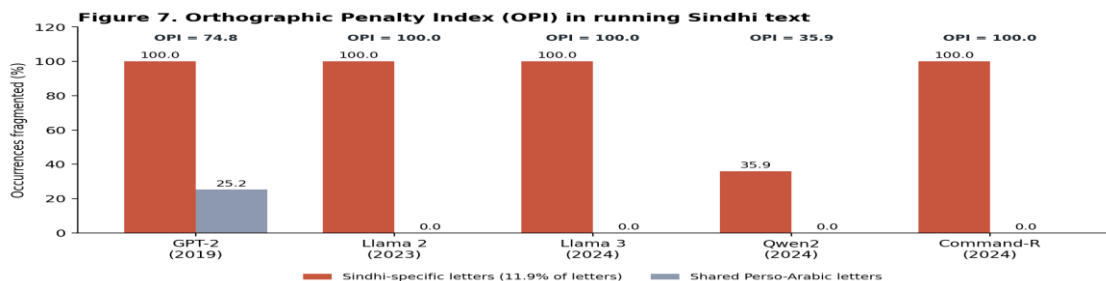


Figure 7. Fragmentation of Sindhi-specific and shared letters in running text, and the resulting OPI

Quantitative interpretation of Figure 7. Sindhi-specific letters make up 11.9% of all letter occurrences in the Sindhi corpus. In Llama 2, Llama 3 and Command-R, 100% of those occurrences were fragmented, while 0% of shared letters were, giving the maximum OPI of 100. GPT-2 fragmented all Sindhi-specific letters and a quarter of shared ones (OPI = 74.8). Qwen2 fragmented 35.9% of Sindhi-specific letter occurrences and almost no shared letters (OPI = 35.9).

Qualitative reading of Figure 7. The OPI isolates a penalty that general measures such as the premium cannot show. Even in the tokenizers where Sindhi's premium is lowest, the language's own letters are systematically broken, while its borrowed letters are not. In SATEF terms, this is inequity at Layer 1, the orthographic foundation, and every higher layer inherits it: each fragmented letter adds a token, and each added token raises cost and shortens context.

Does the premium vary by topic?

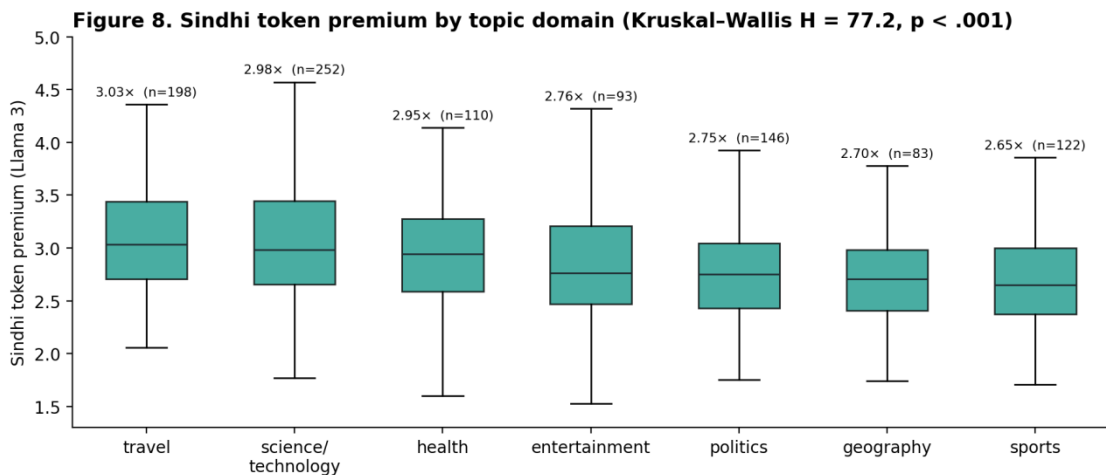


Figure 8. Sindhi token premium (Llama 3) by topic domain

Quantitative interpretation of Figure 8. The Sindhi premium differed significantly across the seven topic domains, Kruskal-Wallis $H = 77.24$, $p < .001$ (Llama 3); the same test on Command-R gave $H = 79.11$, $p < .001$. Median premiums ranged from 3.04 for travel and 2.98 for science and technology down to 2.65 for sports and 2.71 for geography.

Qualitative reading of Figure 8. Travel and science sentences contain many foreign names and technical terms. In English these are compact, often single tokens; in Sindhi they are transliterated into Perso-Arabic script letter by letter and are rarely present in the tokenizer's merges. The domains in which Pakistan most needs AI support, including science education and health, are therefore among the more expensive for Sindhi users.

Unicode variation in digital Sindhi

Figure 9. Unicode variation in digital Sindhi (238 of 1,004 sentences mix heh forms)

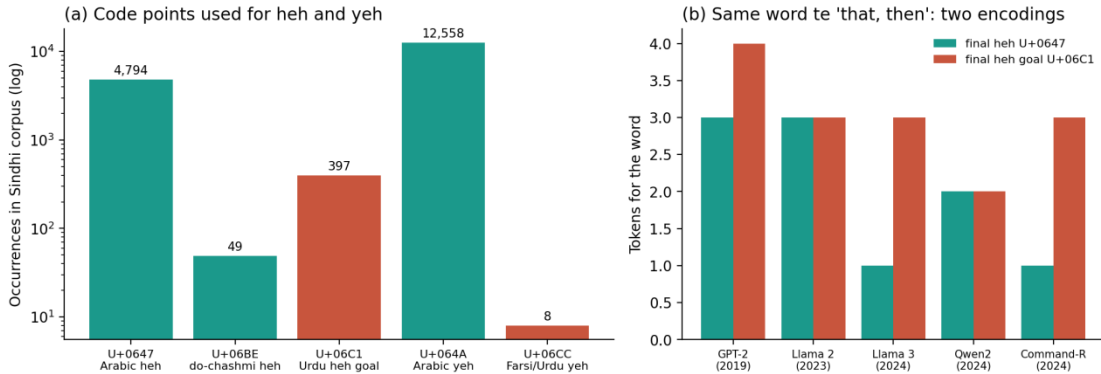


Figure 9. (a) Code points used for heh and yeh in the Sindhi corpus; (b) token cost of the same word written with two different code points

Quantitative interpretation of Figure 9. The Sindhi benchmark text used the Arabic heh (U+0647) 4,794 times, the do-chashmi heh (U+06BE) 49 times, and the Urdu heh goal (U+06C1) 397 times; 238 of the 1,004 sentences mixed the Urdu heh goal with another heh. The Arabic yeh (U+064A) was used 12,558 times and the Farsi/Urdu yeh (U+06CC) 8 times. The same word *te* (اُڻ, “that, then”) cost one token in Llama 3 and Command-R when written with U+0647, but three tokens when written with U+06C1 (اُڻڻ); the same held for *be* (اُڻڻڻ, “also”).

Qualitative reading of Figure 9. Digital Sindhi is not yet orthographically stable. Writers and keyboards mix Sindhi, Urdu and Arabic code points for what readers perceive as the same letter, so a tokenizer sees two different words where a reader sees one (Ali et al., 2019; Unicode Consortium, 2023). This doubles the vocabulary Sindhi needs, splits its statistics, and makes each variant rarer and more fragmented. It should be noted that not all apparent variation is error: the letter ڪ, which represents /k/ in Urdu, is a distinct Sindhi letter for the aspirated /k^h/, and its 998 occurrences in the corpus are legitimate.

Part B: Qualitative Analysis

Segmentation typology (RQ3)

Table 5. Codebook for the qualitative analysis of segmentation

Code	Definition	Example (tokens separated by)	(tokens separated by)
Whole word	The word is a single token	ڇڏي (Llama 3)	
Morph-aligned	Full-character pieces; the last piece is a Sindhi	ڀ ڏ ڇ ڏي (Llama 3)	

	inflectional ending	
Arbitrary sub-word	Full-character pieces not aligned with a known ending	هن آ (Command-R)
Character-level	Every piece is one letter	ي ت (Llama 2)
Byte-fragmented	At least one piece is part of a letter	[byte][byte] ري for ڪري

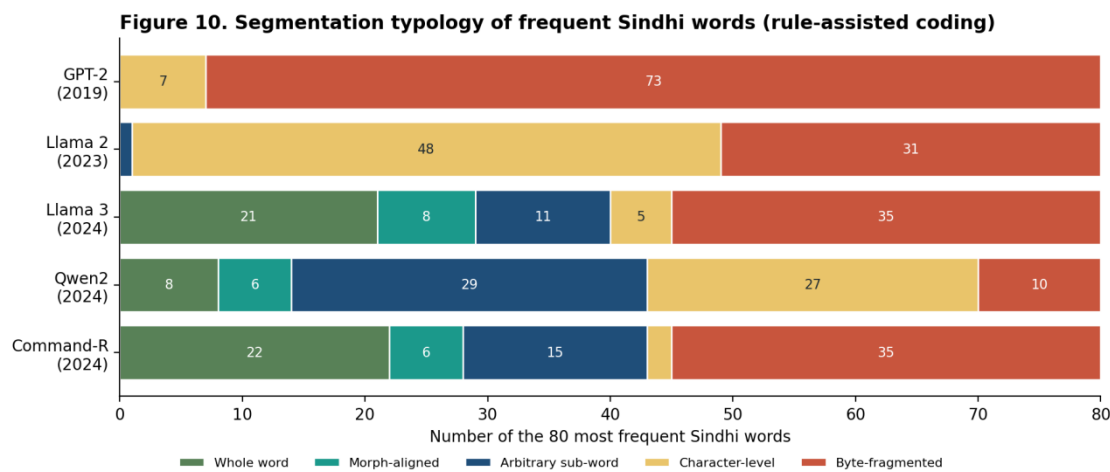


Figure 10. Segmentation typology of the 80 most frequent Sindhi words across tokenizers

Quantitative interpretation of Figure 10 (coding output). Of the 80 most frequent Sindhi words, GPT-2 fragmented 73 into bytes and kept none whole. Llama 2 kept none whole, segmented 48 character by character and byte-fragmented 31. Llama 3 and Command-R kept 21 and 22 words whole, respectively, but still byte-fragmented 35 each. Qwen2 kept 8 whole, byte-fragmented only 10, and segmented most of the rest into single characters (27) or arbitrary pieces (29). For comparison, every one of the 80 most frequent English words was a single token in all five tokenizers. The mean number of tokens per frequent Sindhi word was 4.80 (GPT-2), 3.70 (Llama 2), 2.82 (Llama 3), 2.80 (Qwen2) and 2.84 (Command-R), against exactly 1.00 for English.

Qualitative reading of Figure 10. For English, the most common words of the language are the cheapest units in the model. For Sindhi, even the commonest words are built from fragments. Only a handful of morph-aligned segmentations appear anywhere, which means that token boundaries rarely coincide with the grammatical structure a Sindhi speaker would recognise.

One sentence, five tokenizers

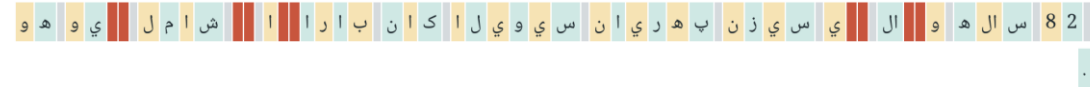
Figure 11. One sentence, five tokenizers (each box = one token; red = byte fragment of a letter; grey = space)

English: 28-year-old Vidal had joined Barça three seasons ago, from Sevilla.

GPT-2 (2019): 61 tokens (English: 19)



Llama 2 (2023): 64 tokens (English: 21)



Llama 3 (2024): 34 tokens (English: 16)



Qwen2 (2024): 36 tokens (English: 17)



Command-R (2024): 36 tokens (English: 17)



Figure 11. A single Sindhi sentence segmented by five tokenizers, with its English equivalent

Quantitative interpretation of Figure 11. The sentence (FLORES id 5) reads, in English, *28-year-old Vidal had joined Barça three seasons ago, from Sevilla*. English needed 16 to 21 tokens. Sindhi needed 61 tokens in GPT-2, 64 in Llama 2, 34 in Llama 3 and 36 in Qwen2 and Command-R.

Qualitative reading of Figure 11. The red blocks mark byte fragments, and in the 2024 tokenizers they fall exactly on the most Sindhi parts of the sentence. In Llama 3 and Command-R, all five fragmented characters are distinctively Sindhi: ڊ in the name وڊال (Vidal), ٽ in ٽي (three), ڪ in ڪا (Barça), ٿ in ٿيو (became), and ڻ (U+06FE), the special Sindhi sign for the postposition *in*. Qwen2 fragments only ٿ. In GPT-2, by contrast, fragmentation spreads to shared letters as well (14 characters). The proper nouns, spelled with shared letters, are split into short sequences of whole letters, while the Sindhi grammatical words are broken below the level of the letter. The figure makes visible what the statistics describe: the cost falls on what is most Sindhi in the sentence.

Qualitative themes

Close reading of the 400 coded segmentations produced five interpretive themes.

Theme 1: The grammar is in fragments. The words most frequently broken into bytes are not rare terms but the core of Sindhi grammar: the light verbs and auxiliaries ڪري (doing), ڪيو (did), ڪئي (did, feminine), ڪي ٿو and ڪي ٿا (present-tense auxiliaries) and the verb ڪرڻ (to do). In GPT-2, Llama 2, Llama 3 and Command-R, ڪرڻ becomes five tokens, four of which are half-letters. A model trained on such input must learn the most basic verbal system of Sindhi from pieces that carry no linguistic meaning on their own (Mielke et al., 2021).

Theme 2: Borrowed efficiency. Where Sindhi words consist only of letters shared with Urdu, Persian and Arabic, the newer tokenizers treat them well: جي, جو and جا (genitive markers), سان (from), ڪان (with), طور (manner) and عام (common) are single tokens in Llama 3 and Command-R. Sindhi’s efficiency is therefore borrowed from its neighbours. It benefits where it overlaps with better-resourced languages and is penalised where it is itself.

Theme 3: The Urdu shortcut. Because Urdu code points are better represented, spelling a Sindhi word with an Urdu letter can make it cheaper, and spelling it with the standard Sindhi letter makes it costlier (Figure 9). The tokenizer thus creates a small but persistent incentive toward Urdu-shaped spelling. In a language whose orthography is already unstable online, such incentives could, over time, reinforce the drift of digital Sindhi toward Urdu conventions, a technological echo of the language hierarchy described by Rahman (1996, 2002).

Theme 4: Morphology out of sight. Sindhi marks infinitives with -ڻ, plural and oblique forms with -ن, and gender and number agreement with -و, -ي and -ا. These endings rarely become token boundaries. The infinitive ending ڻ is itself a Sindhi-specific letter, so it is always split into bytes except where it is absorbed into a longer piece. The agreement paradigm جو, جي, جا appears as three unrelated whole tokens rather than as one stem with three endings. Only occasionally, as in ڀڄي نهجي (one’s own, feminine), does the final token coincide with an agreement ending. A model therefore receives little help from its input in learning Sindhi morphology (Rust et al., 2021).

Theme 5: Coverage is a choice. Qwen2 shows what changes when Sindhi letters are added to the vocabulary: ڙ becomes ڙ | ت, and byte fragmentation almost disappears. But Qwen2’s words are then segmented mostly letter by letter, and its overall premium (2.86) is no lower than that of Llama 3. Adding letters removes the orthographic penalty but not the segmentation penalty; for that, a tokenizer needs Sindhi text in its training data so that frequent Sindhi letter sequences are merged into larger units (Liang et al., 2023; Sennrich et al., 2016).

Discussion

Integrating the strands

Table 6 brings the two strands together, layer by layer, following the logic of the SATEF framework (Creswell & Plano Clark, 2018).

Table 6. Joint display of quantitative and qualitative evidence by SATEF layer

SATEF layer	Quantitative evidence	Qualitative evidence	Integration
1. Orthographic	OPI = 100 in three tokenizers, 74.8 in GPT-2, 35.9 in Qwen2; 238 sentences mix heh code points	Theme 3, the Urdu shortcut; Theme 5, coverage is a choice	Convergent: Sindhi’s own letters are penalised; variation adds further cost

2. Segmentation	Fertility 3.07–5.35 (Sindhi) vs 1.23–1.42 (English); 0–27.5% of frequent Sindhi words whole vs 100% of English	Theme 1, grammar in fragments; Theme 4, morphology out of sight	Convergent: core grammar is fragmented and morphology is rarely visible
3. Economic	Median premium 2.85–4.98×; 766–1,335 Sindhi words per 4,096 tokens vs about 3,300 English	Theme 2, borrowed efficiency	Convergent: Sindhi pays more, and saves only where it overlaps with Urdu
4. Representational	Not measured directly; supported by prior research	All themes	Inferred: fragmented input weakens learning (Rust et al., 2021)

Answering the research questions

RQ1 asked how large the Sindhi token premium is. It is large and universal: a median of 2.85 to 4.98 times the English token count for identical content, with no sentence reaching parity in any tokenizer. This confirms, for a Pakistani language, the global pattern reported by Petrov et al. (2023) and Ahia et al. (2023). Newer tokenizers reduce the premium substantially, from about five to about three, but the reduction has levelled off: Llama 3, Qwen2 and Command-R differ by less than two per cent despite vocabularies ranging from 128,000 to 256,000 tokens. Doubling the vocabulary did not buy Sindhi parity.

RQ2 asked whether Sindhi-specific letters drive fragmentation. They do. The Orthographic Penalty Index reached its maximum of 100 in three of five tokenizers: every occurrence of the 18 Sindhi-specific letters was fragmented, while shared letters were not. This is the study's central empirical contribution. It shows that Sindhi's disadvantage is not only a matter of little data, which is the usual explanation (Joshi et al., 2020), but also of the design of vocabularies around the letters of better-resourced neighbours. Urdu suffers its own orthographic penalties, visible in its high fragmentation in GPT-2 and Command-R, but the letters of the Sindhi alphabet are a distinct and measurable target.

RQ3 asked how frequent Sindhi words are segmented. The qualitative analysis shows that the most frequent grammatical words, especially the light verbs and auxiliaries built on هـ and ڻ , are broken into byte fragments, that Sindhi saves tokens only where its words overlap with Urdu and Arabic, that Urdu-shaped spellings can be cheaper than standard Sindhi ones, and that token boundaries seldom align with Sindhi morphology. These findings extend Rust et al.'s (2021) link between tokenization and model quality by showing *which* linguistic units are lost.

RQ4 asked about implications. They are discussed in Section 5.4.

Theoretical contributions

The study makes three theoretical contributions. First, it moves tokenization-equity research from the level of languages to the level of *scripts within languages*. By separating Sindhi-specific letters from shared ones, the OPI shows that two languages written in the “same” script can be treated very differently, and that the difference is located in identifiable characters. Second, the SATEF framework links orthographic encoding to economic cost and representational quality in a single cascade, providing a vocabulary for discussing AI equity that is precise enough for engineers and accessible enough for policy-makers. Third, the finding that the tokenizer rewards Urdu-shaped spellings connects computational linguistics with sociolinguistics: the prestige hierarchy that Rahman (1996, 2002) and Mansoor (2005) describe in Pakistani institutions reappears inside algorithms, where it can quietly shape how people write.

Practical implications for Pakistan

The findings suggest five practical responses.

1. **Tokenizer audits for Pakistani languages.** Institutions that adopt AI for public services, education or media in Sindh should report the token premium and OPI of the systems they use, in the same way that accessibility standards are reported for websites (Ministry of Information Technology and Telecommunication, 2018).
2. **Vocabulary extension.** Developers of open models used in Pakistan can add the Sindhi-specific letters and frequent Sindhi letter sequences to tokenizer vocabularies, a technique shown to benefit low-resource languages (Liang et al., 2023). Qwen2 demonstrates that including the letters removes most byte fragmentation.
3. **A Unicode normalisation standard for digital Sindhi.** Language authorities, publishers and keyboard developers should agree on normalisation rules for *heh*, *yeh* and related letters, so that the same word is encoded the same way (Ali et al., 2019; Unicode Consortium, 2023).
4. **Sindhi text data.** Because merges are learned from data, open, high-quality Sindhi corpora from newspapers, textbooks and literature are needed so that future tokenizers learn Sindhi words, not only Sindhi letters (Jamro, 2017; Motlani, 2016).
5. **Critical AI literacy in English and linguistics education.** Teachers and students of English and linguistics in Pakistan should understand that AI systems are not language-neutral. The same prompt costs more and fits less in Sindhi, and this knowledge belongs in curricula on language, technology and power (Bender et al., 2021).

Limitations

The study has several limitations. First, tokenizers were reconstructed from published vocabularies and merge rules; although control phrases reproduced the reference token identifiers, small differences in pre-tokenization may remain, particularly for Command-R. Second, FLORES-200 consists of translated Wikipedia-style text, which may differ from spontaneous Sindhi on social media or in speech (Goyal et al., 2022). Third, only five tokenizers were analysed, and commercial systems whose tokenizers are not public could not be included. Fourth, the study measured tokenization, not downstream model quality; the representational layer of SATEF is inferred from prior research rather than tested. Fifth, the morph-aligned code relies on a rule-based list of endings and would benefit from expert morphological annotation. Finally, the analysis covers Sindhi in the Perso-Arabic script used in Pakistan, not Sindhi in Devanagari as used in India.

Conclusion

This study asked what it costs, in the currency of AI, to speak Sindhi rather than English. The answer is precise: about 2.9 to 5.0 times as many tokens for the same content, between one quarter and two fifths of the context window, and a systematic penalty on the very letters that make Sindhi Sindhi. In three of the five tokenizers analysed, every occurrence of the 18 Sindhi-specific letters was broken into byte fragments, while letters shared with Arabic and Urdu were kept whole. Qualitatively, the commonest grammatical words of Sindhi are built from meaningless pieces, its morphology is rarely visible in token boundaries, and Urdu-shaped spellings are rewarded.

The paper's new ideas are the concept of the Sindhi token tax, the Orthographic Penalty Index, and the Script-Aware Tokenization Equity Framework. Together they show that tokenization inequity is not only a consequence of limited data but also a matter of design at the level of the script, and that it can be measured, audited and corrected. The case of Qwen2 shows that change is possible: including Sindhi letters in the vocabulary removes most fragmentation, although parity will require Sindhi text data as well.

For Pakistan, where English holds institutional prestige and Sindhi carries the lives of millions, the lesson is that the old hierarchy of languages is being rebuilt inside new technologies. Linguists, educators, developers and policy-makers can prevent this by treating tokenization as a language-policy question. Future research should measure downstream model quality alongside tokenization, extend the OPI to Punjabi (Shahmukhi), Pashto, Balochi and Saraiki, and test vocabulary-extension methods designed specifically for Pakistani scripts.

References

- Adelani, D. I., Liu, H., Shen, X., Vassilyev, N., Alabi, J. O., Mao, Y., Gao, H., & Lee, E.-S. A. (2024). SIB-200: A simple, inclusive, and big evaluation dataset for topic classification in 200+ languages and dialects. In *Proceedings of the 18th Conference of the European Chapter of*

- the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.
- Ahia, O., Kumar, S., Gonen, H., Kasai, J., Mortensen, D. R., Smith, N. A., & Tsvetkov, Y. (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*. Association for Computational Linguistics.
- Ali, W., Kumar, J., Lu, J., & Xu, Z. (2019). *Word embedding based new corpus for low-resourced language: Sindhi* [Preprint]. arXiv.
- Aryabumi, V., Dang, J., Talupuru, D., Dash, S., Cairuz, D., Lin, H., Venkitesh, B., Smith, M., Campos, J. A., Tan, Y. C., Marchisio, K., Bartolo, M., Ruder, S., Locatelli, A., Kreutzer, J., Frosst, N., Gomez, A., Blunsom, P., Fadaee, M., Üstün, A., & Hooker, S. (2024). *Aya 23: Open weight releases to further multilingual progress* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2405.15032>
- Bender, E. M., Gebru, T., McMillan-Major, A., & Shmitchell, S. (2021). On the dangers of stochastic parrots: Can language models be too big? In *Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency* (pp. 610–623). <https://doi.org/10.1145/3442188.3445922>
- Blasi, D., Anastasopoulos, A., & Neubig, G. (2022). Systematic inequalities in language technology performance across the world's languages. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics.
- Braun, V., & Clarke, V. (2006). Using thematic analysis in psychology. *Qualitative Research in Psychology*, 3(2), 77–101. <https://doi.org/10.1191/1478088706qp0630a>
- Cohen, J. (1988). *Statistical power analysis for the behavioral sciences* (2nd ed.). Lawrence Erlbaum.
- Creswell, J. W., & Plano Clark, V. L. (2018). *Designing and conducting mixed methods research* (3rd ed.). SAGE.
- Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., et al. (2024). *The Llama 3 herd of models* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2407.21783>
- Friedman, M. (1937). The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the American Statistical Association*, 32(200), 675–701. <https://doi.org/10.1080/01621459.1937.10503522>
- ggml-org. (2024). *llama.cpp* [Computer software]. GitHub. <https://github.com/ggml-org/llama.cpp>
- Goyal, N., Gao, C., Chaudhary, V., Chen, P.-J., Wenzek, G., Ju, D., Krishnan, S., Ranzato, M., Guzmán, F., & Fan, A. (2022). The Flores-101 evaluation benchmark for low-resource and multilingual machine translation. *Transactions of the Association for Computational Linguistics*, 10, 522–538. https://doi.org/10.1162/tac1_a_00474
- Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, 9(3), 90–95. <https://doi.org/10.1109/MCSE.2007.55>
- Hussain, S. (2008). Resources for Urdu language processing. In *Proceedings of the 6th Workshop on Asian Language Resources*. Asian Federation of Natural Language Processing.

- Jamro, W. A. (2017). Sindhi language processing: A survey. In *2017 International Conference on Innovations in Electrical Engineering and Computational Technologies (ICIEECT)*. IEEE.
- Joshi, P., Santy, S., Budhiraja, A., Bali, K., & Choudhury, M. (2020). The state and fate of linguistic diversity and inclusion in the NLP world. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics* (pp. 6282–6293). Association for Computational Linguistics.
- Karpathy, A. (2023). *llama2.c* [Computer software]. GitHub. <https://github.com/karpathy/llama2.c>
- Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 66–75). Association for Computational Linguistics.
- Kudo, T., & Richardson, J. (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* (pp. 66–71). Association for Computational Linguistics.
- Liang, D., Gonen, H., Mao, Y., Hou, R., Goyal, N., Ghazvininejad, M., Zettlemoyer, L., & Khabsa, M. (2023). XLM-V: Overcoming the vocabulary bottleneck in multilingual masked language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.
- Limisiewicz, T., Balhar, J., & Mareček, D. (2023). Tokenization impacts multilingual language modeling: Assessing vocabulary allocation and overlap across languages. In *Findings of the Association for Computational Linguistics: ACL 2023*. Association for Computational Linguistics.
- Mansoor, S. (2005). *Language planning in higher education: A case study of Pakistan*. Oxford University Press.
- McKinney, W. (2010). Data structures for statistical computing in Python. In S. van der Walt & J. Millman (Eds.), *Proceedings of the 9th Python in Science Conference* (pp. 56–61). <https://doi.org/10.25080/Majora-92bf1922-00a>
- Mielke, S. J., Alyafeai, Z., Salesky, E., Raffel, C., Dey, M., Gallé, M., Raja, A., Si, C., Lee, W. Y., Sagot, B., & Tan, S. (2021). *Between words and characters: A brief history of open-vocabulary modeling and tokenization in NLP* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2112.10508>
- Ministry of Information Technology and Telecommunication. (2018). *Digital Pakistan policy*. Government of Pakistan.
- Motlani, R. (2016). Developing language technology tools and resources for a resource-poor language: Sindhi. In *Proceedings of the NAACL Student Research Workshop*. Association for Computational Linguistics.
- NLLB Team, Costa-jussà, M. R., Cross, J., Çelebi, O., Elbayad, M., Heafield, K., Heffernan, K., et al. (2022). *No language left behind: Scaling human-centered machine translation* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2207.04672>

- Pakistan Bureau of Statistics. (2023). *7th population and housing census 2023*. Government of Pakistan.
- Petrov, A., La Malfa, E., Torr, P. H. S., & Bibi, A. (2023). Language model tokenizers introduce unfairness between languages. In *Advances in Neural Information Processing Systems 36 (NeurIPS 2023)*.
- Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., & Sutskever, I. (2019). *Language models are unsupervised multitask learners* [Technical report]. OpenAI.
- Rahman, T. (1996). *Language and politics in Pakistan*. Oxford University Press.
- Rahman, T. (2002). *Language, ideology and power: Language-learning among the Muslims of Pakistan and North India*. Oxford University Press.
- Ranathunga, S., & de Silva, N. (2022). Some languages are more equal than others: Probing deeper into the linguistic disparity in the NLP world. In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing*. Association for Computational Linguistics.
- Rust, P., Pfeiffer, J., Vulić, I., Ruder, S., & Gurevych, I. (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)* (pp. 3118–3135). Association for Computational Linguistics.
- Sennrich, R., Haddow, B., & Birch, A. (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)* (pp. 1715–1725). Association for Computational Linguistics.
- Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., et al. (2023). *Llama 2: Open foundation and fine-tuned chat models* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2307.09288>
- Unicode Consortium. (2023). *The Unicode standard, version 15.1.0*. The Unicode Consortium.
- Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., et al. (2020). SciPy 1.0: Fundamental algorithms for scientific computing in Python. *Nature Methods*, 17(3), 261–272. <https://doi.org/10.1038/s41592-019-0686-2>
- Wilcoxon, F. (1945). Individual comparisons by ranking methods. *Biometrics Bulletin*, 1(6), 80–83. <https://doi.org/10.2307/3001968>
- Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., et al. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 38–45). Association for Computational Linguistics.
- Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., & Raffel, C. (2022). ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10, 291–306. https://doi.org/10.1162/tacl_a_00461

Yang, A., Yang, B., Hui, B., Zheng, B., Yu, B., Zhou, C., et al. (2024). *Qwen2 technical report* [Preprint]. arXiv. <https://doi.org/10.48550/arXiv.2407.10671>