SINTECHCOM: Science, Technology and Communication Journal
Vol. 7, No. 1, October 2026, pp. 269-280, DOI: 10.59190/stc.v7i1.423

# HUP-TIS: A hybrid unicode-property tokenizer with adaptive template instruction stripping for word-limit validation of Indonesian research proposals

**Yuvi Darmayunata*, Yogi Yunefri, Lucky Lhaura Van FC**
Department of Informatics, Universitas Lancang Kuning, Pekanbaru 28266, Indonesia

## ABSTRACT

Research-grant information systems enforce section-level word limits on proposals, but the methods behind these checks are rarely validated. This study audited word-limit validation in the Research and Community Service Information System (SIPP) of Universitas Lancang Kuning on 30 real proposal PDFs from one submission cycle. The original code of two production generations was executed end-to-end: one counts words with the PHP function str_word_count(), the other with a Unicode-property regular expression. Both were compared with hybrid unicode-property tokenizer with adaptive template instruction stripping (HUP-TIS), a proposed pipeline that anchors section headings with layout cues and aligns lines sequentially with the official template instructions. Segmentation quality was measured without ground truth from the provenance of word 5-grams, and each system's count error was decomposed exactly into segmentation, instruction-leakage, and tokenizer components. The main finding, a failure mode not described in the literature we reviewed, is instruction-induced boundary drift. Template instructions that name other sections attracted the heading patterns of both production systems. As a result, the text counted as methods was introduction content in 28 to 29 documents, and 69 to 76% of methods and expected results sections were flagged as over quota, against 0 to 6.7% under HUP-TIS. Segmentation caused 83.5 to 85.9% of the absolute count error and the tokenizer only 1.9 to 4.8%. The newer system, despite its better tokenizer, disagreed with HUP-TIS on more quota decisions than the legacy system (91 vs 57). HUP-TIS processed a PDF in a median of 97 ms, against about 650 ms for the production systems. The results are limited to one institution and one submission cycle; word-level validation against human annotation is ongoing.

## ARTICLE INFO

**Article history:**

Received Sep 7, 2026
Revised Oct 12, 2026
Accepted Oct 19, 2026
Published Oct 31, 2026

**Keywords:**

Instruction Stripping
Research Proposal
Text Segmentation
Tokenization
Word-Limit Validation

**Copyright (c) 2026:**

Yuvi Darmayunata, Yogi Yunefri, Lucky Lhaura Van FC

**License:**

This work is licensed under a Creative Commons Attribution 4.0 International License (CC BY).

*Corresponding Author
E-mail address: yuvidarmayunata@unilak.ac.id

## 1. INTRODUCTION

Universities rely on web-based information systems to manage internal research grants, and administrative screening is the first gate a proposal must pass. At Universitas Lancang Kuning, the instructions printed in the proposal template limit the Title to 20 words, the Summary to 300 words, the Keywords to five items, the Introduction to 1,000 words (1,500 in the Applied scheme), and the Methods to 1,000 words [1]. The institutional grant guideline adds a 500-word limit for the Expected Results. Neither document defines what a word is, so screening outcomes depend entirely on how the system implements the check.

The Research and Community Service Information System (SIPP) has two production generations. The legacy system, sipp_lama, counts words with the PHP function str_word_count() [2], which ignores digits. Its successor, SIPP v3, counts matches of a Unicode-property regular expression. Be-

fore counting, both systems must cut the text extracted from the PDF into the template sections and discard the instructions printed in the template. Discussions of word-count accuracy usually focus on the tokenizer, although what counts as a word has long been recognized as a design decision of tokenization itself [3,4]. Whether tokenization or segmentation dominates the error in practice has not been measured. The two generations therefore share the same segmentation logic and differ mainly in character normalization, tokenizer, and keyword handling, whereas HUP-TIS changes segmentation and instruction removal themselves (Table 1).

Indonesian morphology, including affixation, hyphenated reduplication, and clitics, has been studied extensively for stemming and lemmatization [5,6], and Indonesian NLP benchmarks such as IndoNLU [7] and IndoLEM [8] have matured [9]. General-purpose tokenizers such as NLTK [10] and spaCy [11] are widely used, and preprocessing choices are known to change downstream results [12,13]. Text extraction from PDF is itself error-prone, and extraction tools differ markedly in how well they preserve words and reading order [14,15]. Document layout analysis exploits typography and position to recover document structure [16-19], and tools such as GROBID and CERMINE segment scholarly articles [20,21]. These lines of work, however, target English articles or downstream understanding tasks. None addresses fill-in proposal forms whose template instructions remain in the text and mention the names of other sections, and none evaluates quota decisions.

HUP-TIS (Hybrid Unicode-Property Tokenizer with Adaptive Template Instruction Stripping), the pipeline proposed here, differs from these approaches in three respects. First, its target is a fill-in form rather than a finished article: the template instructions remain in the submitted text, are typeset much like body text, and name other sections, so a heading detector that ignores them anchors on the wrong line. Second, it treats the official template as a known prior and aligns each body line sequentially with the instruction text of its section, instead of learning section labels from annotated data; GROBID and CERMINE rely on models trained on scholarly articles, whereas HUP-TIS needs only the template file and a small development audit (six documents here). Third, it is evaluated on the administrative outcome, the over-quota decision for each section, rather than on extraction or labeling accuracy alone. The novelty therefore lies not in layout cues or instruction removal taken separately, but in their combination: typographic cues locate the candidate headings, and sequential alignment with the official instructions decides which lines between those headings are template text, so that an instruction that names another section can neither anchor a heading nor be counted as content. On the tokenization side, HUP-TIS does not propose a new linguistic tokenizer. Its counter implements an explicit administrative protocol (Table 2) that states which tokens count as words, because the grant documents leave this undefined, and general-purpose tokenizers are evaluated against this protocol rather than replaced by it.

This study answers four research questions on 30 real proposals. RQ1: where do production word-limit validators fail, in tokenization, segmentation, or instruction leakage? RQ2: how do the quota decisions of sipp_lama and SIPP v3 differ from those of layout- and template-aware segmentation? RQ3: which tokenization phenomena occur in real proposals, and how do tokenizers handle them? RQ4: what does each approach cost computationally? The contributions are (i) HUP-TIS, a hybrid pipeline that combines layout cues with sequential alignment against template instructions; (ii) two ground-truth-free measures, segmentation purity and an exact error decomposition, for auditing deployed validators; (iii) the identification of instruction-induced boundary drift, the central finding of this study, a failure mode in which template instructions that name other sections attract the heading patterns of a validator, in two production systems; and (iv) a corpus-based taxonomy of tokenization phenomena.

## 2. RESEARCH METHODS

All stages (Figure 1) were executed within a single Jupyter notebook using a fixed random seed and recorded library versions to support reproducibility. Rules were designed and audited exclusively on the development split, keeping rule refinement separate from evaluation. As a reproducibility check, HUP-TIS was executed as a single end-to-end function for each PDF and reproduced the stage-wise word counts across all 180 evaluated sections, confirming consistency between the two execution approaches.

**Table 1.** Production systems and HUP-TIS compared by pipeline component.

| Component | sipp_lama | SIPP v3 | HUP-TIS |
|---|---|---|---|
| PDF text extraction | Smalot PdfParser, plain text | Smalot PdfParser, plain text | PyMuPDF, lines with font size, bold/italic, and position |
| Character normalization | None | Spaces, quotes, dashes | Full Unicode cleaning and NFKC (Section 2.3) |
| Section boundaries | First match of unanchored, case-insensitive patterns | Literal markers if present; otherwise as sipp_lama | Whole-line headings (Eq. 1), typographic levels, order A to H enforced |
| Template instructions | Not removed | Only "Petunjuk:" blocks and fixed placeholders removed | Sequential alignment with the official instruction text; italic fallback |
| Word tokenizer | str_word_count() (digits ignored) | Unicode-property pattern (Eq. 2) | Protocol in Table 1 |
| Keywords | Words; limit 50 | Items split by semicolon, comma, or line break; limit 5 | Items split by semicolon; limit 5 |
| Quota | Fixed per scheme | Fixed per scheme | Parsed from the document's instruction; scheme mode as fallback |

**Table 2.** Operational protocol for counting words.

| Phenomenon | Rule | Example and count |
|---|---|---|
| Template instructions, placeholders | Not counted | …… dst = 0 |
| Tables, diagrams, captions | Not counted | Tabel 2. … = 0 |
| Sub-heading labels | Label 0, title counted | D1. Latar Belakang = 2 |
| List enumerators, bullets | Not counted | (1), a., • = 0 |
| Numbered citations | Not counted | [1], (2) = 0 |
| Stand-alone punctuation | Not counted | =, & = 0 |
| Numerals, ranges, percentages | One word | 2019-2025 = 1 |
| Reduplication, hyphenated words | One word | masing-masing = 1 |
| Slash compounds | One word | dan/atau = 1 |
| Abbreviations, underscores, URLs | One word each | dkk., karbon_aktif = 1 |
| Keywords section | Unit = item (split by ;) | a; b; c = 3 |

### 2.1. Corpus

Thirty proposal PDFs submitted to SIPP in the April 2026 cycle of the internal grant schemes were used with the institution's permission to access the SIPP archive: 25 Fundamental, 3 Junior Lecturer (Dosen Pemula), and 2 Applied (4 to 33 pages; mean 10.0, SD 5.9). Each file received an anonymous identifier from its SHA-256 hash. No file was duplicated, scanned, or encrypted. The corpus was split, stratified by scheme, into a development set of six documents (about 20% of each scheme: five Fundamental and one Junior Lecturer; neither Applied proposal could be spared) and a test set of 24 documents. The development set was used only to design and audit the HUP-TIS rules, whereas the production systems were run unchanged on every document. Results are therefore reported on all 30 documents, so that every production failure is counted wherever it occurs. Thirty documents yield 240 headings and 180 evaluated sections, and the drift counts, decision disagreements, and error decomposition reported below are exact counts over these sections rather than sampled estimates. The corpus is thus adequate to show that a failure mode exists and how often it occurs within this cycle, but not to estimate its rate at other institutions (Section 3.7).

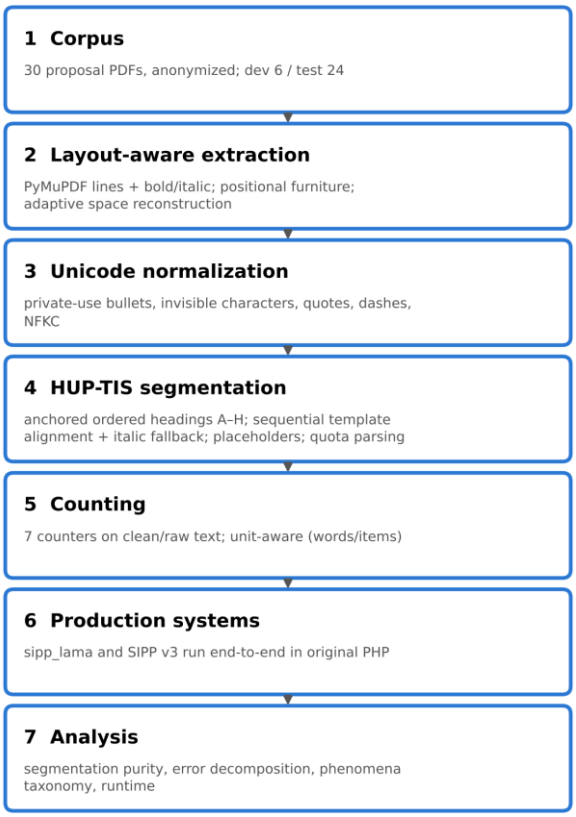



**Figure 1.** Experimental pipeline.

The 30 proposals were sampled so that each was submitted by a different researcher, which varies writing and formatting habits across the corpus instead of repeating the style of a few authors. The sample serves as a test collection: in routine screening, administrators check proposals one at a time, and a sample of this size makes every section of every document traceable in the analysis. The development set keeps about 80% of each scheme unseen and covers the Fundamental and Junior Lecturer templates; the Applied scheme, with only two proposals, could not contribute a development document. HUP-TIS found all 192 headings of the 24 test documents in order, which indicates that its rules were not fitted to the development documents.

### 2.2. Layout-Aware Extraction

Text was extracted line by line with PyMuPDF [22], keeping font size, the proportion of bold and italic characters, and position. Spans within a line were concatenated without inserted spaces; the resulting token counts matched plain PyMuPDF extraction exactly on all 30 files. Page furniture was defined positionally: only page numbers or repeated lines inside the top or bottom 7% of the page were removed. An earlier frequency-only rule wrongly removed 1,028 lines (7.7%), mostly list enumerators and table cells. One PDF stored no space characters, so its title was read as a single word by every method. Character geometry showed intra-word gaps of -0.047 to -0.085 em and word gaps of +0.092 to +0.099 em. A space was therefore inserted when a gap exceeded the line's median gap by 0.12 em, applied only to lines containing a token of at least 20 letters. This changed two lines in the corpus and restored the title from 1 to 13 words.

### 2.3. Unicode Normalization

Private-use bullets were replaced by spaces, invisible characters were removed, special spaces, curly quotes, and dash variants were mapped to ASCII, and NFKC (Normalization Form Compatibility Composition) normalization [23] was applied. Normalization changed 5.8% of lines but only -0.05% of tokens. Ligatures (*modifikasi*, *dijanjikan*) and an inserted space (*dius ulkan*) originated in the official template instructions, and the ellipsis U+2026 (754 occurrences in 29 documents) came from template placeholders that applicants did not delete.

### 2.4. HUP-TIS Segmentation

Headings were candidates only if they occupied a whole line, as defined by the pattern in Equation (1). Candidates were accepted in levels, first exact capitals and bold, then bold, then plain, and a lower level was searched only between already-accepted neighboring headings, which enforces the order A to H.

$$^([A-H])\s*\.\s*(JUDUL|RINGKASAN|KATA\ KUNCI|PENDAHULUAN|METODE|HASIL\ YANG\ DIHARAPKAN|JADWAL\ PENELITIAN|DAFTAR\ PUSTAKA)\$ \quad (1)$$

Each body line was reduced to a key (NFKC, lower case, digits replaced by #, letters only), which is robust to ligatures, inserted spaces, and scheme-specific numbers. A line was treated as an instruction if (a) its key continued the position reached in the official instruction text of that section, allowing a gap of 40 key characters, with the first line requiring at least 25 characters; (b) its key of at least 40 characters occurred anywhere in that instruction; or (c) it was italic, not bold, and at least 60% of its words belonged to the instruction vocabulary, which covers template variants. Bullet-only lines did not break an instruction block, captions never counted as instructions, and placeholder lines (dots or "dst") were removed. The quota was parsed from the document's own instruction; if the applicant had deleted it, the modal quota of the same scheme was used. The sequential rule was introduced after the development audit showed that unordered matching discarded 186 tokens of applicants' sub-headings (e.g., *Latar Belakang*, *Metode Penelitian*) in 44 sections. The bullet rule removed 74 leaked instruction tokens of the Junior Lecturer template.

### 2.5. Counting Methods and Protocol

Seven counters were compared: str_word_count() and the SIPP v3 pattern in Equation (2), both executed on PHP 8.4.23; whitespace split; NLTK 3.10.3 word_tokenize [10]; the rule-based Indonesian tokenizer of spaCy 3.8.16 [11]; a reduplication-aware regular expression; and the HUP-TIS counter. Each counter received *clean* input (HUP-TIS content lines) and *raw* input (all text between two headings), except the HUP-TIS counter, which was applied to clean input only; six counters were therefore run on raw input. The HUP-TIS counter implements the protocol in Table 2, excludes caption, list-marker, and lone-number lines, and counts keywords as items separated by semicolons.

$$[\p{L}\p{N}]+(?:[.,'`&/-][\p{L}\p{N}]+)* \quad (2)$$

### 2.6. Production Systems and Analysis Measures

Table 1 summarizes how the two production generations of SIPP and HUP-TIS differ. Both production systems locate a section with case-insensitive patterns whose alternatives include the bare section name (for example, METODE), take the first match anywhere in the text, and end the section at the first subsequent match of the next pattern. SIPP v3 additionally normalizes characters, removes instruction blocks that begin with the label "Petunjuk:" together with fixed placeholder sentences, and uses literal section markers when a document contains them. Neither system removes instructions that lack these labels, which is the case in the template used in this cycle.

The original PHP classes of both systems, with Smalot PdfParser [24] and CodeIgniter 4, were run on each PDF with its scheme, as in deployment, and their counted text was reconstructed for analysis. Segmentation purity maps every word 5-gram of a system's section text to its origin under HUP-TIS (a section's content, template instructions, or front matter). Purity is the share originating from the same section, coverage is the share of that section's content that is included, and boundary drift occurs when the majority originates elsewhere. Error decomposition splits the difference between a system count $C$ and the HUP-TIS count $H$ using the system's own tokenizer $T$ (Equation 3), where $P$ is the system's section text, $R$ the raw text, and $K$ the clean text. The four terms sum exactly to the total.

$$C - H = [C - T(P)] + [T(P) - T(R)] + [T(R) - T(K)] + [T(K) - H] \quad (3)$$

The four terms are the internal residual, segmentation plus extraction, instruction leakage, and tokenizer or protocol effects. Because the terms are evaluated in this order, the decomposition is exact

but order-dependent: a tokenizer difference on leaked instruction text, for example, is attributed to leakage. Each term is therefore read as the error a stage adds given the preceding stages, not as an independent cause. Runtime was measured as the median of three (PHP) or five (HUP-TIS) repetitions per document, and tokenizer cost was measured inside each runtime. Because H is produced by HUP-TIS rather than by human annotation, the decomposition shows where a production system departs from HUP-TIS; it measures error only to the extent that HUP-TIS itself is correct (Section 3.7).

Human annotation has a separate role: it validates word-level counts, which purity and decomposition cannot do. Two annotators work on neutral sheets that contain every line between headings without any method output, so the reference cannot inherit HUP-TIS decisions. Annotator 1 has covered all 30 documents (61,253 tokens) and annotator 2 a stratified sample of 10 test documents (20,452 tokens); agreement will be measured with ICC(2,1) [25,26], and disagreements will be adjudicated into the final reference. The adjudicated reference will provide three checks that the present analysis cannot: the word-count error of every system, including HUP-TIS, in each section; the accuracy of each over-quota decision; and the share of the decision disagreements in Section 3.2 that are errors of the production systems rather than of HUP-TIS. The identification of boundary drift does not depend on this reference, because drift is established from the origin of the counted text.

## 3. RESULTS AND DISCUSSIONS

### 3.1. Segmentation and Tokenizer Effects (RQ1)

HUP-TIS found all 240 headings (30 documents by 8 sections) in order, 239 at the strictest level. Template instructions were separated in 28 to 30 of 30 documents per section; one applicant had deleted all instructions, and the quotas parsed from instructions agreed within each scheme (Applied: Introduction 1,500). Table 3 contrasts the effect of the tokenizer with that of instruction leakage. On identical clean text, median counts differed by only 3% to 7% between tokenizers, and quota decisions changed for at most two documents per section. Keeping the instructions raised the share of Titles over quota from 0% to 40% to 60%, because the Title instruction alone adds 6 to 7 tokens. Word-based counters flagged 83% to 100% of Keywords sections against the five-item limit, whereas counting items gave a median of exactly 5.

**Table 3.** Sections flagged as over quota by the seven counters on clean text and six counters on raw text (HUP-TIS counter excluded) (n = 30 per section).

| Section | Limit | Clean text (%) | Raw text (%) | Added tokens (median) |
|---|---|---|---|---|
| Title | 20 | 0 | 40 – 60 | 6 – 7 |
| Summary | 300 | 3 – 7 | 10 – 13 | 16 – 17 |
| Keywords | 5 items | 0 (items); 83 – 100 (words) | 100 | 9 – 11 |
| Introduction | 1,000/1,500 | 3 – 7 | 3 – 13 | 40 – 44 |
| Methods | 1,000 | 7 – 13 | 17 – 27 | 80 – 82 |
| Expected results | 500 | 0 | 0 | 10 |

### 3.2. Production Systems (RQ2)

Table 4 and Figure 2 compare the two production systems with HUP-TIS. Both systems located the Methods section at the word *metode* inside the Summary instruction ("…urgensi, tujuan, metode, dan luaran…") and the Expected Results section at the phrase *hasil yang diharapkan* inside the Methods instruction. As a result, the text counted as Methods was Introduction content (median purity 0%) in 29 (sipp_lama) and 28 (SIPP v3) documents, and the text counted as Expected Results was Methods content in 26 and 25 documents. The Introduction was truncated (coverage 62% to 66%). In one document, an Introduction that exceeded its limit (1,010 of 1,000 words under HUP-TIS) passed because it was truncated to 459 to 473 words. Neither system found the Introduction of the document whose instructions had been deleted, and SIPP v3 found none of sections A to F in one further document.

Instruction leakage changes quota decisions through two mechanisms. The first is additive: instruction tokens inside a correctly bounded section are counted as content. It can only create false

flags; the Title instruction alone adds 6 to 7 tokens to a section with a median of 14 words under HUP-TIS, enough to flag 40% to 62% of Titles in the production systems against 0% under HUP-TIS. The second mechanism is displacive and far larger: an instruction that names another section supplies the match for that section's heading pattern, so the boundary moves by a whole block of text. Because the Summary instruction contains *metode*, the Methods section starts inside the Summary, ends inside the Methods instruction, and consists mostly of Introduction content. This mechanism produces both false flags (Methods 69% to 70% against 7%; Expected Results 73% to 76% against 0%, with a median of 155 words under HUP-TIS against 814 to 874 counted) and false passes (the truncated Introduction above). Whether leaked text changes a decision therefore depends less on its length than on whether it names another section.

**Table 4.** Production systems versus HUP-TIS: median count, sections flagged over quota, median segmentation purity, and documents with boundary drift.

| Section | Median count, (L / v3 / H) | Over quota, % (L / v3 / H) | Purity, % (L / v3) | Drift (L / v3) |
|---|---|---|---|---|
| Title | 20 / 21 / 14 | 40 / 62 / 0 | 79 / 79 | 0 / 0 |
| Summary | 241 / 246 / 225 | 10 / 14 / 3 | 94 / 94 | 0 / 0 |
| Keywords | 19 / 6 / 5 | 3 / 93 / 0 | 50 / 50 | 15 / 14 |
| Introduction | 552 / 524 / 770 | 0 / 4 / 3 | 89 / 89 | 2 / 2 |
| Methods | 1,106 / 1,118 / 651 | 70/ 69 / 7 | 0 / 0 | 29 / 28 |
| Expected results | 814 / 874 / 155 | 73 / 76 / 0 | 16 / 17 | 26 / 25 |

L = sipp_lama; v3 = SIPP v3 (29 documents, because no section was found in one document; 28 for the Introduction, which was also not found in the document whose instructions had been deleted); H = HUP-TIS. sipp_lama applies a 50-word limit to Keywords, whereas SIPP v3 and HUP-TIS apply five items. Percentages are rounded to whole numbers; for example, Methods under HUP-TIS is 2 of 30 documents (6.7%).

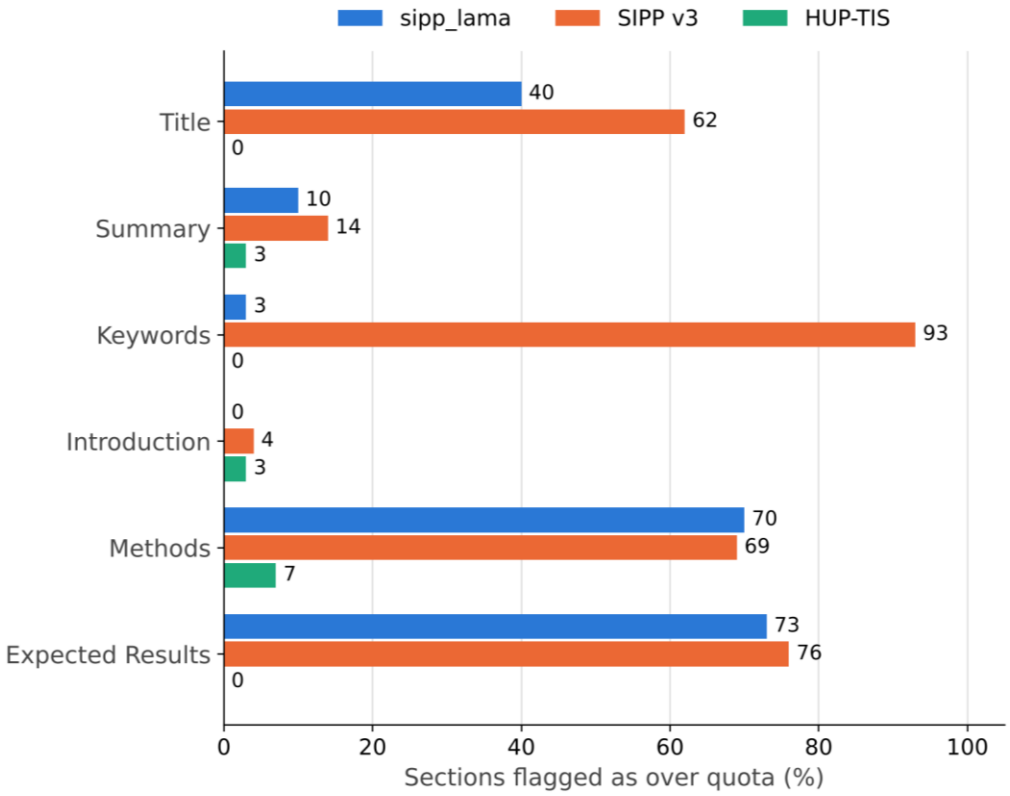



**Figure 2.** Sections flagged as over quota by each system.

SIPP v3 disagreed with HUP-TIS on 91 quota decisions, sipp_lama on 57. The improvements of SIPP v3 backfired under leakage: its regular expression counts the digits in "maksimal 20 kata" within the Title instruction, and its keyword-item splitter turned the instruction fragment "Isian 5 kata kunci… (;)" into a sixth item, flagging 93% of Keywords sections. A better tokenizer without better segmentation therefore produced more wrong decisions. These are disagreements with HUP-TIS, not verified errors; in sections with boundary drift, however, the counted text demonstrably belongs to another section.

### 3.3. Error Decomposition (RQ1)

Over 352 sections with counts from both sides, segmentation plus extraction accounted for 83.5% (sipp_lama) and 85.9% (SIPP v3) of the absolute count error, instruction leakage for 9.2% and 9.3%, and the tokenizer for 1.9% and 4.8% (Figure 3). By section, Title errors came almost entirely from leakage (91%), whereas Introduction, Methods, and Expected Results errors came from segmentation (78% to 98%), with mean signed effects of about -310, +345, and +640 words, respectively. The internal residual of sipp_lama (5.3%) stems from two sections whose reconstructed text did not match the text counted internally. Eight sections that a system did not find at all are reported separately.

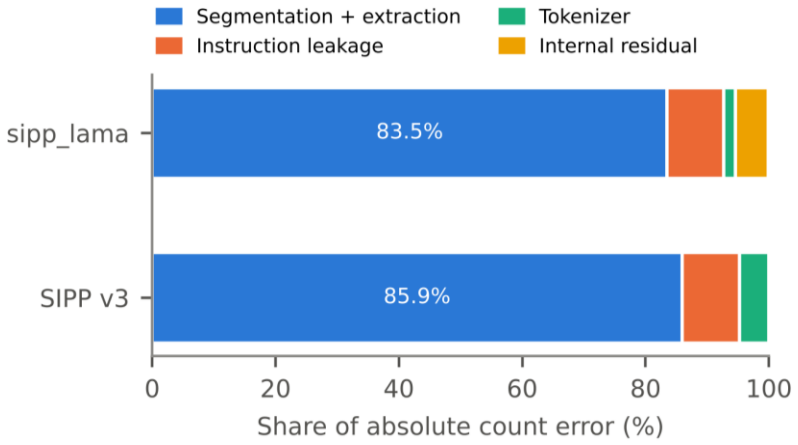



**Figure 3.** Decomposition of absolute count error (internal residual: 5.3% for sipp_lama, 0.0% for SIPP v3).

Segmentation dominates because of the scale at which each error operates. A tokenizer disagreement affects single tokens, occurs in only 2.5% to 3.5% of tokens (Section 3.4), and partly cancels out. A segmentation error moves a boundary by a whole block. For Expected Results, the displaced boundary adds roughly 660 to 720 words to a median section of 155 words, whereas the tokenizer changes the same section by about four to five words. The shares are therefore shares of error magnitude, not of wrong decisions: Title errors are small in words yet change the decision for 40% to 62% of Titles, because the limit is only 20 words. Two qualifications apply: the segmentation term includes extraction differences between Smalot PdfParser and PyMuPDF, and the shares depend on the order of terms in Equation (3). Improving the tokenizer of a validator therefore cannot repair its decisions while its section boundaries are wrong, as the SIPP v3 results show.

### 3.4. Tokenization Phenomena (RQ3)

Table 5 classifies the 56,673 content tokens of sections A, B, D, E, and F and counts each tokenizer's absolute deviation from the protocol. The premise that str_word_count() splits reduplicated words is not supported: reduplication occurred 105 times (0.19% of tokens) and was always counted as one word, because the function treats hyphens as word characters. Its real errors are ignored numerals and split slash compounds, whereas modern tokenizers count citations and list enumerators. Every tokenizer deviated on only 2.5% to 3.5% of tokens, and their ranking depends on the definition of *word*, which the grant documents do not provide.

**Table 5.** Tokenization phenomena and absolute deviation from the protocol (tokens).

| Phenomenon | Frequency | str_word_count() | SIPP v3 | NLTK | spaCy |
|---|---|---|---|---|---|
| Numerals | 1,097 | 875 | 71 | 50 | 16 |
| List enumerators | 709 | 176 | 625 | 625 | 625 |
| Numbered citations | 575 | 9 | 575 | 575 | 584 |
| Hyphenated words | 410 | 27 | 4 | 1 | 397 |
| Stand-alone punctuation | 390 | 0 | 0 | 0 | 0 |
| Slash compounds | 134 | 140 | 2 | 0 | 143 |
| Reduplication | 105 | 0 | 0 | 0 | 15 |
| Sub-heading labels | 99 | 99 | 99 | 99 | 99 |
| Numeric ranges | 52 | 0 | 0 | 0 | 52 |
| URLs | 24 | 65 | 25 | 25 | 0 |
| All content tokens; total deviation (%) | 56,673 | 2.6% | 2.5% | 2.5% | 3.5% |

Whitespace split deviated on 3.1% of tokens, penalized mainly by enumerators, citations, and punctuation.

### 3.5. Computational Cost (RQ4)

End-to-end, sipp_lama and SIPP v3 needed a median of 653 and 654 ms per PDF (95th percentile about 1.35 s), and HUP-TIS needed 97 ms (236 ms). Tokenizers alone cost 9 µs (whitespace), 13 µs (str_word_count), 30 µs (SIPP v3 pattern), 158 µs (spaCy), and 552 µs (NLTK) per section, so tokenization is never the bottleneck. The speed advantage of HUP-TIS comes mainly from the PDF parser (C-based PyMuPDF versus pure-PHP Smalot); for this reason, HUP-TIS will be integrated into SIPP as a Python microservice rather than ported to PHP.

Processing is independent per document, so cost grows linearly: at the median of 97 ms, 1,000 proposals take about 1.6 min on one process (about 4 min at the 95th percentile), and documents can be processed in parallel. These figures extrapolate runtimes measured on 30 PDFs, so a load test over a full submission cycle remains part of the integration. Manual effort grows with the number of template versions, not documents (Section 3.6). In the planned integration, the CodeIgniter 4 application of SIPP changes at one point: on upload, it sends the PDF and its scheme to the HUP-TIS service through a REST interface and receives, for each section, the count, the quota, the decision, and the lines removed as instructions, which the applicant can inspect. The check runs once per upload and adds well under one second. Because administrators currently verify proposals one at a time, the per-section output, including the lines removed as instructions, is intended to support their check rather than replace it: an administrator can see why a section was flagged and confirm the decision in the document.

### 3.6. Adaptation to Other Institutions and Templates

HUP-TIS separates generic components from template-specific inputs. Layout-aware extraction, Unicode normalization, space reconstruction, the italic fallback for instructions, and the counting protocol do not depend on the institution. Only three inputs are template-specific: the section headings and their order (Equation 1), the official instruction text of each section, and the quota rule. Adapting HUP-TIS to another institution or template year therefore requires supplying these inputs from the template file and a short development audit, such as the six-document audit used here; no retraining is needed, because HUP-TIS learns nothing from annotated data. Templates that do not italicize instructions, or that number sections differently, need these inputs checked rather than assumed. Purity and error decomposition need no ground truth and can audit any deployed validator before HUP-TIS is adopted.

### 3.7. Limitations

The main limitation is scope: the corpus comes from one institution and one submission cycle and is dominated by the Fundamental scheme; it contains no Professor-scheme proposal, so the rates reported here should not be generalized to other institutions or template years without a new audit. The

mechanism of drift, however, applies wherever unanchored heading patterns meet instructions that name other sections; only its frequency is template-specific.

Second, HUP-TIS serves as the reference for purity and decomposition. The identification of drift does not depend on this choice, because it is measured from the origin of the text and the instruction 5-grams that a system counted as content can be traced to the official template text. The magnitudes do depend on it: any segmentation error of HUP-TIS itself is attributed to the production system, so the shares in Figure 3 include that error. HUP-TIS found all 240 headings in order, which limits but does not remove this risk. Consequently, outside drifted sections the decision counts in Section 3.2 are disagreements rather than verified errors, and the word-level superiority of HUP-TIS cannot be claimed until the human reference is complete. Third, word-level accuracy against the double-annotated ground truth is still being produced (Section 2.6); it will quantify the residual error of HUP-TIS itself. The HUP-TIS counter implements the protocol by design, so its deviation in Table 5 is zero by construction. The instruction dictionary was taken from the Fundamental template; other templates rely on the typographic fallback. Memory use of the PHP processes was not measured.

## 4. CONCLUSION

The central finding of this study is instruction-induced boundary drift: in both production generations of SIPP, template instructions that mention other sections attracted the unanchored heading patterns, so most Methods and Expected Results sections were measured on the wrong text and flagged as over quota. Word-limit validation therefore failed mainly at segmentation, not tokenization: segmentation accounted for more than 83% of the count error and the tokenizer for less than 5%, and the improved tokenizer of SIPP v3 increased decision errors because instructions still leaked into the counted text. HUP-TIS, which anchors headings with layout cues and aligns lines sequentially with the template instructions, segmented every heading of the 30 proposals and runs in about 0.1 s per PDF. For institutions that validate proposals automatically, the lesson is to audit section boundaries before refining the tokenizer, which the ground-truth-free purity measure makes possible. Instruction-induced boundary drift is the main contribution of this study: because it arises from the interaction between template instructions and unanchored heading patterns, it can affect any validator that checks fill-in templates in this way, regardless of the language or the tokenizer used. These results come from one institution and one submission cycle. Future work will report word-level accuracy against human annotation, extend the corpus to other schemes, institutions, and template years, and integrate HUP-TIS into SIPP as a microservice.

## AUTHOR CONTRIBUTIONS

Yuvi Darmayunata: Conceptualization, methodology, HUP-TIS development, software implementation, formal analysis, and writing—original draft. Yogi Yunefri: Investigation, data curation, execution of production-system benchmarks, performance evaluation, and writing—review and editing. Lucky Lhaura Van FC: Validation, comparative analysis, interpretation of findings, visualization, and writing—review and editing. All authors contributed to discussing the results, revising the manuscript, and approving the final version.

## FUNDING STATEMENT

This research received no external funding.

## DECLARATION OF AI USE

The authors declare that no AI tools were used in preparing this manuscript.

## CONFLICT OF INTEREST

The authors declare no conflicts of interest.

## DATA AVAILABILITY STATEMENT

All data supporting this study are included in the article. No separate datasets are available.

## ETHICAL APPROVAL

Ethical approval was unnecessary, as no humans or animals were involved.

## ACKNOWLEDGMENTS

This research was funded by the Institute for Research and Community Service (LPPM) Universitas Lancang Kuning under the Fundamental Institutional Grant Scheme (APBU), Fiscal Year 2026.

## REFERENCES

[1] LPPM Universitas Lancang Kuning. (2026). *Format usulan proposal penelitian fundamental: Isian substansi proposal*. Universitas Lancang Kuning.
URL: https://lppm.unilak.ac.id

[2] PHP Group. (2026). *str_word_count: Return information about words used in a string*. PHP Manual.
URL: https://www.php.net/manual/en/function.str-word-count.php

[3] Webster, J. J., & Kit, C. (1992). Tokenization as the initial phase in NLP. *Proceedings of the 14th International Conference on Computational Linguistics (COLING 1992)*, **4**, 1106-1110.
URL: https://aclanthology.org/C92-4173

[4] 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. *arXiv preprint arXiv:2112.10508*.
DOI: https://doi.org/10.48550/arXiv.2112.10508

[5] Abidin, Z., Junaidi, A., & Wamiliana. (2024). Text stemming and lemmatization of regional languages in Indonesia: A systematic literature review. *Journal of Information Systems Engineering and Business Intelligence*, **10**(2), 217-231.
DOI: https://doi.org/10.20473/jisebi.10.2.217-231

[6] Rizki, A. Z., Tjahyanto, A., & Trialih, R. (2019). Comparison of stemming algorithms on Indonesian text processing. *TELKOMNIKA*, **17**(1), 95-102.
DOI: https://doi.org/10.12928/TELKOMNIKA.v17i1.10183

[7] Wilie, B., Vincentio, K., Winata, G. I., Cahyawijaya, S., Li, X., Lim, Z. Y., Soleman, S., Mahendra, R., Fung, P., Bahar, S., & Purwarianti, A. (2020). IndoNLU: Benchmark and resources for evaluating Indonesian natural language understanding. *Proceedings of AACL-IJCNLP 2020*, 843-857.
DOI: https://doi.org/10.18653/v1/2020.aacl-main.85

[8] Koto, F., Rahimi, A., Lau, J. H., & Baldwin, T. (2020). IndoLEM and IndoBERT: A benchmark dataset and pre-trained language model for Indonesian NLP. *Proceedings of COLING 2020*, 757-770.
DOI: https://doi.org/10.18653/v1/2020.coling-main.66

[9] Aji, A. F., Winata, G. I., Koto, F., Cahyawijaya, S., Romadhony, A., Mahendra, R., Kurniawan, K., Moeljadi, D., Prasojo, R. E., Baldwin, T., Lau, J. H., & Ruder, S. (2022). One country, 700+ languages: NLP challenges for underrepresented languages and dialects in Indonesia. *Proceedings of ACL 2022*, 7226-7249.
DOI: https://doi.org/10.18653/v1/2022.acl-long.500

[10] Bird, S., Klein, E., & Loper, E. (2009). *Natural language processing with Python*. O'Reilly Media.
URL: https://tjzhifei.github.io/resources/NLTK.pdf

[11] Honnibal, M., Montani, I., Van Landeghem, S., & Boyd, A. (2020). spaCy: Industrial-strength natural language processing in Python. *Zenodo*.
DOI: https://doi.org/10.5281/zenodo.1212303

[12] Hickman, L., Thapa, S., Tay, L., Cao, M., & Srinivasan, P. (2022). Text preprocessing for text mining in organizational research: Review and recommendations. *Organizational Research Methods*, **25**(1), 114-146.
DOI: https://doi.org/10.1177/1094428120971683

[13] Meidelfi, D., Yulherniwati, Rahmayuni, I., Hidayat, T., & Chandra, D. (2025). Impact of pre-processing on Indonesian extractive text summarization. *G-Tech: Jurnal Teknologi Terapan*, **9**(4), 2311-2321.
DOI: https://doi.org/10.33379/gtech.v9i4.7211

[14] Bast, H., & Korzen, C. (2017, June). A benchmark and evaluation for text extraction from PDF. *2017 ACM/IEEE joint conference on digital libraries (JCDL)*, 1-10.
DOI: https://doi.org/10.1109/JCDL.2017.7991564

[15] Meuschke, N., Jagdale, A., Spinde, T., Mitrović, J., & Gipp, B. (2023, March). A benchmark of pdf information extraction tools using a multi-task and multi-domain evaluation framework for academic documents. *International Conference on Information*, 383-405.
DOI: https://doi.org/10.1007/978-3-031-28032-0_31

[16] Zhong, X., Tang, J., & Jimeno Yepes, A. (2019). PubLayNet: Largest dataset ever for document layout analysis. *Proceedings of the International Conference on Document Analysis and Recognition (ICDAR)*, 1015-1022.
DOI: https://doi.org/10.1109/ICDAR.2019.00166

[17] Li, M., Xu, Y., Cui, L., Huang, S., Wei, F., Li, Z., & Zhou, M. (2020). DocBank: A benchmark dataset for document layout analysis. *Proceedings of COLING 2020*, 949-960.
DOI: https://doi.org/10.18653/v1/2020.coling-main.82

[18] Xu, Y., Li, M., Cui, L., Huang, S., Wei, F., & Zhou, M. (2020). LayoutLM: Pre-training of text and layout for document image understanding. *Proceedings of KDD 2020*, 1192-1200.
DOI: https://doi.org/10.1145/3394486.3403172

[19] Huang, Y., Lv, T., Cui, L., Lu, Y., & Wei, F. (2022). LayoutLMv3: Pre-training for document AI with unified text and image masking. *Proceedings of the 30th ACM International Conference on Multimedia*, 4083-4091.
DOI: https://doi.org/10.1145/3503161.3548112

[20] Lopez, P. (2009). GROBID: Combining automatic bibliographic data recognition and term extraction for scholarship publications. *Research and Advanced Technology for Digital Libraries (ECDL 2009)*, 473-474.
DOI: https://doi.org/10.1007/978-3-642-04346-8_62

[21] Tkaczyk, D., Szostek, P., Fedoryszak, M., Dendek, P. J., & Bolikowski, Ł. (2015). CERMINE: Automatic extraction of structured metadata from scientific literature. *International Journal on Document Analysis and Recognition*, **18**(4), 317-335.
DOI: https://doi.org/10.1007/s10032-015-0249-8

[22] Artifex Software. (2026). *PyMuPDF documentation*. Artifex Software
URL: https://pymupdf.readthedocs.io

[23] Davis, M., & Whistler, K. (2026). *Unicode normalization forms (Unicode Standard Annex #15)*. The Unicode Consortium.
URL: https://www.unicode.org/reports/tr15/

[24] Smalot. (2026). *PdfParser: PHP library to parse PDF files and extract elements like text*. GitHub.
URL: https://github.com/smalot/pdfparser

[25] Shrout, P. E. & Fleiss, J. L. (1979). Intraclass correlations: Uses in assessing rater reliability. *Psychological Bulletin*, **86**(2), 420-428.
DOI: https://doi.org/10.1037/0033-2909.86.2.420

[26] Koo, T. K., & Li, M. Y. (2016). A guideline of selecting and reporting intraclass correlation coefficients for reliability research. *Journal of Chiropractic Medicine*, **15**(2), 155-163.
DOI: https://doi.org/10.1016/j.jcm.2016.02.012