Bachelor's Thesis Computing Science

Language Identification Challenges in Low-Resource Web Data

Evaluating and adapting Resiliparse for low-resource languages

Marten Bosma
s1041706

June 15, 2026

First supervisor/assessor:
Djoerd Hiemstra

Second assessor:
Arjen de Vries

Radboud University

Abstract

Web indexes are an essential component of the infrastructure of search engines, which we rely on daily. The Open Web Index is a European open-source web index pilot that aims to provide transparent, safe, sovereign and open access to the internet for Europe. The web index is built on a web indexing pipeline that includes language identification.

The language identification for the Open Web Index is done through a tool, called Resiliparse. This tool is required to perform very efficient and reliable language identification of web pages. Achieving this requirement on web-extracted text is not an arbitrary task, especially for minority and low-resource languages. This research aims to evaluate the performance of Resiliparse on web-extracted data for low-resource languages.

The methodology compares Resiliparse, FastText, and UrlExtractor across WiLI-2018, CommonLID, a controlled Germanic-language dataset, and manually annotated OWI web slices. It then analyses errors associated with web noise, text length, linguistic similarity, confidence scores, and training-domain mismatch. Finally, we evaluate cutoff-, similarity-, and URL-based fallback approaches by comparing their accuracy and runtime with the baseline systems.

The results show that Resiliparse consistently performs faster where FastText is consistently more accurate. Furthermore, the results suggest that several factors are closely associated with the decreased performance of the tool including web noise, linguistic similarity, text length and training data mismatch. Several adaptations show promise even while keeping runtime low but need further evaluation. The research contributes to the existing body of research through its analysis of Resiliparse in an OWI-related setting. It also suggests that further work should focus on establishing larger corpora of annotated web-extracted data and the improvement of language identification tools for the web.

Contents

1 Introduction **3**

2 Preliminaries **5**
2.1 Language Resource Levels . . . 5
2.2 Language Identification . . . 5
2.3 N-grams and Tokenization . . . 6
2.4 Cavnar-Trenkle method . . . 6
2.5 Reservoir Sampling . . . 6
2.6 Challenges of Web Data . . . 7
2.7 Evaluation Metrics . . . 7

3 Research **8**
3.1 Research Design . . . 8
3.2 Resiliparse Evaluation Module . . . 8
3.3 Datasets and Data Preparation . . . 9
3.3.1 WiLI-2018 . . . 9
3.3.2 OWI Slice . . . 9
3.3.3 Germanic Language Cluster . . . 11
3.3.4 CommonLID . . . 11
3.4 Evaluated Language Identification Systems . . . 11
3.4.1 Resiliparse . . . 12
3.4.2 FastText Model . . . 12
3.4.3 UrlExtractor . . . 13
3.5 Baseline Evaluation . . . 13
3.5.1 Discussion Baseline Evaluation . . . 15
3.6 Error Factor Analysis . . . 16
3.6.1 Mixed Annotation Analysis . . . 16
3.6.2 Annotation Disagreements . . . 17
3.6.3 Text-Length . . . 17
3.6.4 Discussion Text-Length . . . 18
3.6.5 Linguistic Similarity . . . 19
3.6.6 Discussion Linguistic Similarity . . . 21
3.6.7 Cutoff Sweep . . . 22
3.6.8 Discussion Cutoff Sweep . . . 23
3.6.9 Discussion Training Data and Domain Mismatch . . . 24
3.7 Adapted LID Approaches . . . 24
3.7.1 URL/Domain-Based Adaptation . . . 24

3.7.2 Discussion URL/Domain-Based Adaptation . . . 25
3.7.3 Language-Aware FastText Trigger . . . 26
3.7.4 Discussion Language-Aware FastText Trigger . . . 27
3.7.5 Comparative Performance and Runtime . . . 28
3.7.6 Discussion Comparative Performance and Runtime . . . 31

4 Related Work **33**
4.1 Language Identification Methods . . . 33
4.2 Short Text, Web Noise, and Domain Mismatch . . . 34
4.3 Low-Resource and Closely Related Languages . . . 34
4.4 Benchmarks and Web-Domain Evaluation . . . 35
4.5 Metadata and Lightweight Adaptation . . . 35
4.6 Position of This Thesis . . . 36

5 Conclusions **37**
5.1 Summary of Findings . . . 37
5.2 Practical Implications for the OWI . . . 38
5.3 Reflections on Validity and Limitations . . . 38
5.4 Recommendations and Future Work . . . 39
5.5 Overall Conclusion . . . 39

A Technical Implementation **44**
A.1 Source Code Availability . . . 44
A.2 CLI and Package Structure . . . 44
A.3 Critical Implementation Snippets . . . 45
A.3.1 Runtime Measurement and Output Capture . . . 45
A.3.2 Input Validation and Dataset Balancing . . . 46
A.3.3 Routing Strategies and Adapted Systems . . . 49
A.3.4 Metrics Computation and Runtime Analysis . . . 52
A.4 Data Schemas . . . 55
A.4.1 OWI Slice Files . . . 55
A.4.2 OWI Slice Language Composition . . . 55
A.4.3 GLC Layout . . . 55
A.4.4 Detector Output CSV . . . 56
A.4.5 Publication Visual Inputs . . . 56

B Supplementary Results **57**
B.1 Reference Tables . . . 57
B.2 Resource-Level Supplement . . . 60
B.2.1 Text-Length Controls . . . 60
B.3 Cutoff Sweep Controls . . . 61
B.4 Language-Similarity Diagnostics . . . 61
B.5 Runtime Details . . . 63
B.6 Score-Gap Diagnostics . . . 65

Chapter 1

Introduction

Search engines have become an essential part of modern life, as we rely on them to access the internet and find the information we are looking for. Through continuous crawling, processing and indexing web content, these search engines continue to retrieve the most relevant results to our search queries. The effectiveness of a search engine at retrieving the most relevant results, is largely dependent on the quality and coverage of the web index underneath the hood of the search engine. The search engine's web index is responsible for sorting and categorising the web pages then the search engine ranks these indexed web pages to return the most relevant results to the search query.

Building a web index requires continuous crawling through the internet, processing web pages and categorising their contents. The computational power, storage capacity and underlying infrastructure needed for this scale of data is considerable. These requirements results in a small number of large tech companies maintaining such web indexes. The concentration of these services within these large tech companies creates a dependency on these organisations to access the web.

To address some of these concerns, the European Union (EU) has funded the Open Web Search (OWS) intitiative which has developed the Open Web Index (OWI). The OWS aims to provide free, open and unbiased access to information. The OWI is the result of that and is freely available to researchers, public institutions and organisations to develop their own search and analysis systems [17, 16]. Like any other web index, the OWI relies on crawling, processing and categorisation of web content.

The categorisation of web content makes use of language identification (LID), which helps support retrieval of web data in the user's language. In order to perform LID, the OWI uses Resiliparse, a tool which handles extraction and parsing of web data. Due to the multilingual nature of the internet, correctly classifying the language of each web page is essential in ensuring effective indexing and retrieval. Ensuring reliable LID across all supported languages is therefore an important requirement for the OWI.

One of the major challenges within natural language processing (NLP) is the availability of reliable, accurarely labelled data. This challenge becomes particularly apparent when we move towards multilingual and web-related research, where data quality and coverage varies strongly across languages [9, 12]. As a result, research within the field has focused towards high-resource languages, with larger datasets and extensive literature digitally available. The relatively scarcer resources for lower-resource languages has led them to be less researched. Additionally, web data poses another unique challenge in its heterogeneous nature. Web pages have varying text length, levels of noise and can include multiple languages within a single web page [13, 18].

Shorter text segments are especially challenging for language identification systems [3, 22]. All of these characteristics can negatively impact performance of the LID systems, which within a web indexing pipeline propagates through the pipeline as LID is a foundational step.

Resiliparse is one such system, serving as a component within OWI's web indexing pipeline. In the version evaluated in this thesis, Resiliparse reports support for 101 languages through its language-detection implementation and supported language profiles [7]. As a lightweight system, it is designed for multilingual LID within a web-scaled environment. Resiliparse needs to handle both the challenges of web-extracted data and the lower-resources available for many of the languages it supports. The combination of these two challenges requires further study particularly with the constraints of a web-scale environment. Therefore, this thesis evaluates and analyses the LID capabilities of Resiliparse within these contraints, with the purpose of identifying potential improvements.

The primary objective of this thesis is to evaluate the effectiveness of the language identification component employed by Resiliparse for lower-resource languages in web-extracted text. To achieve this, this thesis assesses the accuracy of Resiliparse across a range of lower-resource languages, investigates the factors that influence language identification performance, including linguistic similarity, data characteristics, and web-specific noise, and examines potential approaches for improving classification performance while maintaining the computational efficiency required for large-scale web indexing systems such as the Open Web Index.

This thesis aims to address the following main research question:

> *How effective is Resiliparse's language identification method for low-resource languages in web-extracted data?*

Our contributions are as follows:

- We provide an empirical evaluation of Resiliparse's language identification performance on web-extracted data, showing the performance gap between clean Wikipedia-based benchmarks and real-world web content for lower-resource languages.
- We develop a research-oriented command-line interface (CLI) that enables reproducible large-scale evaluation and analysis of language identification systems.
- We construct and release a human-annotated web-extracted evaluation dataset, designed to add to the existing body of web-extracted evaluation data.
- We perform a systematic cutoff sweep analysis over the evaluated datasets to investigate the relationship between confidence thresholds and classification performance across languages and datasets.
- We propose and evaluate a similarity-aware fallback trigger that leverages linguistic similarity information and improves language identification performance in the evaluated lower-resource and closely related languages while keeping low runtime overhead.

This thesis is organized as follows. Chapter 2 introduces the language-identification concepts and evaluation metrics used throughout the work. Chapter 4 positions the thesis within previous work on language identification, noisy web data, low-resource languages and metadata signals. Chapter 3 describes the datasets, experiments, error analysis and adapted approaches, while Chapter 5 discusses the findings.

Chapter 2

Preliminaries

This chapter introduces the fundamental concepts and definitions on which the work in this thesis is based. These preliminaries provide the terminology needed to understand the later research design, experiments, and results. The chapter first introduces the definition of language resource levels, language identification as a task, then explains character n-grams and the Cavnar-Trenkle profile method, followed by reservoir sampling, the challenges of web data and the metrics used throughout the thesis.

2.1 Language Resource Levels

The terms high-resource, mid-resource, and low-resource describe differences in the data, tools, and institutional support available for languages. These categories do not have one universally accepted definition and can change with the task, domain, and type of resource being considered [15]. This thesis therefore uses an explicit operational definition suited to the written-data setting of language identification.

Resource level is defined using the approximate number of articles in each language's Wikipedia. Languages with fewer than 100,000 articles are classified as low-resource, languages with at least 100,000 but fewer than 1,000,000 articles as mid-resource, and languages with at least 1,000,000 articles as high-resource.

2.2 Language Identification

Language identification (LID) is a classification problem where a text is provided and a language label needs to be assigned. For written text, this usually means deciding which natural language best describes a document, sentence, or fragment of text. LID can be approached with statistical profile methods, supervised classifiers, or neural and subword-based models, and the suitability of each approach depends on the setting in which it is used [9].

For this thesis, four properties are particularly important: efficiency, language coverage, training-data requirements, and robustness to noisy input. A web-indexing pipeline needs to process large amounts of text, so an accurate method that is too slow may still not be the optimal choice. At the same time, low-resource languages often have less labelled training data and a smaller digital presence, which makes broad language coverage and robustness especially important.

2.3 N-grams and Tokenization

Tokenization is the process of splitting text into smaller units, called tokens. These units can be words, subwords, characters, bytes, or sequences of such units. An n-gram is a sequence of n adjacent tokens. In this thesis, when referring to n-grams we mean the character-size n-gram as it works well for profile-based LID.

For example, the word `Hello` can be represented as character unigrams H, e, l, l, and o. With boundary markers, its character bigrams can be written as $\_H$, He, el, ll, lo, and $o\_$. Character n-grams are useful for LID because languages differ in common spelling patterns, affixes, letter combinations, and character sequences. This makes them effective even when texts are too short for reliable word-level evidence or when spelling mistakes are made.

2.4 Cavnar-Trenkle method

As Resiliparse's language identification method is based on the Cavnar-Trenkle method, it is important to understand how this method works. The Cavnar-Trenkle method is a statistical profile method for text categorization based on character n-grams [6]. For language identification, the method first builds one profile per language. Each profile is made up of the most frequent n-grams observed in the training data, retaining only the highest-ranked n-grams and ordering them by frequency. When a document needs to be classified, we create a ranked n-gram profile for it.

The document profile is then compared with each language profile using an out-of-place (OOP) score. For each n-gram in the document profile, the method checks where that n-gram appears in the language profile. If it appears in a similar rank position, the penalty is small. If it appears far away, or does not appear at all, the penalty is larger. The language with the lowest total OOP score is treated as the closest match.

This can be written as $D(T, L) = \sum_{g \in T} |r_T(g) - r_L(g)|$, where $r_T(g)$ is the rank of n-gram g in the text profile and $r_L(g)$ is the rank of the same n-gram in the language profile. If an n-gram from the text profile is missing from the language profile, the method assigns a maximum penalty for that n-gram.

Cavnar and Trenkle use only the highest-ranked n-grams for comparison, arguing that the most frequent n-grams carry the strongest category signal while lower-ranked n-grams increasingly reflect topic-specific or domain-specific content [6]. This is important for the later experiments because a low OOP score should indicate a close profile match, while small score gaps between top-ranked languages can indicate ambiguity.

2.5 Reservoir Sampling

To create manageable and reproducible evaluation subsets of web pages from the OWI crawl data, this thesis uses reservoir sampling. Reservoir sampling is a method for extracting a fixed-size random sample from a stream or collection when the full size of the input may not be known in advance [23]. The method keeps a reservoir of selected items. The first (k) items fill the reservoir, where (k) is the desired sample size. Every later item is considered against the current reservoir with a probability that decreases as more items have already been seen, replacing an existing reservoir item when selected. This makes it possible to sample from large data streams without storing every candidate item.

In this thesis, reservoir sampling is used to create fixed-size OWI slices from larger crawl outputs. Using a fixed random seed makes the sample reproducible: the same input order,

sample size, and seed produce the same slice again. This matters because the OWI crawl is too large to inspect manually in full, while the resulting annotated slices still need to be traceable and repeatable.

2.6 Challenges of Web Data

Web data is more difficult for language identification than clean benchmark text. Web pages vary strongly in length: some contain longer coherent paragraphs, while others contain only short fragments, labels, menus, or image-heavy content. Short text segments are known to make language identification harder because they provide less evidence for the classifier [22, 3].

Web-extracted text can also include boilerplate, navigation menus, advertisements, corrupted text, or mixed-language content, where boilerplate and extraction artifacts are a known problem in web-corpus construction[18], and multilingual web-crawled corpora can contain serious quality issues such as incorrect language labels, uneven coverage, and noisy text[13]. These characteristics are directly relevant to this thesis because the evaluated OWI data consists of web-extracted text and may therefore contain the same forms of noise, mixed-language content, and uneven language evidence.

2.7 Evaluation Metrics

Language identification performance is evaluated using accuracy, precision, recall, and F1-score. Accuracy measures the proportion of instances for which the predicted language equals the ground truth. Precision measures how often predictions for a language are correct, while recall measures how many instances of a language are successfully identified. The F1-score combines precision and recall into a single metric.

This thesis uses macro averaging continuously for understanding performance. With macro averaging, metrics are calculated per language and then averaged, giving each language equal weight regardless of how many examples it has. This is important for low-resource evaluation because a model can appear strong under frequency-weighted metrics while still performing poorly on smaller or less represented languages. Confusion matrices are also used to show which languages are correctly classified and which languages are most commonly confused with each other.

Chapter 3

Research

3.1 Research Design

This chapter and the research consists of three stages that build on each other, with the findings of each stage informing the experiments conducted in the next stage. The first stage of experiments establishes a baseline and aims to validate an assumption about Resiliparse's LID within the OWI pipeline. The assumption based on anecdotal evidence is that Resiliparse performs significantly worse on a low-resource language like Frisian. In the second stage we perform error analysis on the results from the first stage. By looking at the misclassifications we aim to identify contributing causes in general and focus on low-resource languages for the use case. In the third stage we take these leading causes to adapt Resiliparse with the goal of improving performance on low-resource languages.

3.2 Resiliparse Evaluation Module

In order to effectively perform the research within the thesis, we have created a set of scripts that would extract, analyse and visualise all of the data. These scripts, in essence, perform all of the steps taken throughout the research, resulting in a set of tables and figures fit for analysis. These scripts were then bundled into one module, called `rsp`, with commands for reproducibility along with detailed steps outlined in a documentation file. The module contains, extractors, detectors and a number of scripts responsible for processing of data produced by the models. The module, along with all of the steps required to reproduce the research described below, is made available within a repository.

We operationalize the baseline through the generated detector-output file `rp_outputs.csv` for each dataset. Each row corresponds to one evaluated document and contains the gold label, the Resiliparse rank predictions and OOP scores, the FastText prediction and score information, the UrlExtractor prediction where available, text length and detector runtime measurements. This row-level format is important because the same output file supports both metrics and later error-factor analyses. The `rsp` CLI is used to keep enrichment, detector-output generation, sweeps, and publication tables reproducible from the same artifacts.

3.3 Datasets and Data Preparation

In this research an assortment of datasets and slices is used. The datasets were selected to separate three evaluation conditions: clean Wikipedia-style benchmark data, manually corrected web data, and a controlled cluster of linguistically similar Germanic languages. This separation is important because a low score on web data alone would not show whether the failure is caused by linguistic similarity, noisy extraction, short text, or a mismatch between training and evaluation domains.

3.3.1 WiLI-2018

The WiLI-2018 dataset is used as a clean benchmark. It consists of 235,000 paragraphs covering 235 languages and was derived from Wikipedia data [20, 21]. The dataset provides a useful baseline since Wikipedia-style paragraphs are relatively clean, well-structured, and close to the kind of data used to construct many language profiles. For this research the test portion is used as an external benchmark, with labels normalized into the ISO language-code format used by the rsp module.

WiLI-2018 is not meant to simulate the OWI production setting. Instead, it is a reference point for what Resiliparse can achieve when the text is well structured and when low-resource languages are evaluated in a cleaner domain. This is necessary because the claim is not merely that low-resource languages are difficult to classify, but that the gap between clean data and web data is large enough to matter for web indexing.

The source WiLI-2018 files are not modified, and the generated `rp_outputs.csv` contains results for the full benchmark. The filtering is applied only during analysis and figure generation. At that stage, we retain entries whose gold language label also occurs in at least one of the other evaluation datasets: CommonLID, GLC, or an OWI Slice. Together, these datasets contain 77 languages that are also present in WiLI-2018. The full WiLI test split contains 116,500 rows over 230 languages, with 500 rows per language. After filtering, 39,000 rows over these 77 languages remain. This analysis subset is referred to as `WiLI_Filtered` in the figures and generated tables.

3.3.2 OWI Slice

Two new datasets were made for this research. The first we call OWI Slice, which contains data from the Open Web Index developed within the OpenWebSearch.eu initiative [17, 16]. A full crawl from the OWI was taken (`Index-main.owi@it41-2026-02-03:2026-02-03`), which is a routinely performed crawl by a part of the OWS initiative using a seed. The crawl consists of web pages of which the main body of the content is extracted and indexed by language. Three slices were taken from this crawl: a random slice, a Dutch slice and a Frisian slice, each containing 100 web pages. These slices were obtained through the use of Owilix, a tool suggested for this purpose by OWI and a simple bash command that utilises the fixed reservoir sampling we discussed before.

The three slices serve different purposes. The Frisian slice is the main target slice because it represents the low-resource language for which the performance gap was expected. The Dutch slice is included as a related higher-resource Germanic comparison point. The random slice provides a broader web-data control because it contains pages without selecting for a single target language. The slices were sampled with a fixed reservoir-sampling procedure so that the selection is reproducible while still being independent of document order in the crawl output, making the slice construction repeatable and expandable for future work.

These web pages have varying text length and noise and may have been misclassified by the original indexing pipeline. The OWI Slice data was therefore not treated as immediately reliable

ground truth. The raw documents were cleaned and converted into Label Studio tasks. This step prepares annotation inputs, but it does not by itself produce an evaluation dataset. The language-specific slices are also limited to true positives and false positives within the selected OWI language index. As finding false negatives across the full crawl would require searching documents indexed under other languages and is outside the scope of this dataset.

The random and Frisian slice were annotated independently by two annotators with help from Google Translate when the language was outside our expertise. The Frisian and Dutch slices are neither completely in the language that is to be expected, with the Frisian OWI Slice containing only 42 pages with Frisian content. This contamination is itself an important observation: the OWI language index can contain pages whose actual language differs from the indexed language, and therefore the evaluation must be based on corrected human labels. For the slices with two annotators, the annotation exports were compared and disagreements were resolved unanimously. The Dutch slice has only one annotator export, and the annotation should therefore be read with that additional limitation in mind. The annotators are not professional language experts, and Google Translate was used for more than ten samples outside our direct language knowledge. Mixed-language pages were labelled as mixed during annotation. These rows are excluded from use in language identication as the label mixed cannot be compared against, a limitation caused by the chosen annotation method.

Table 3.1 shows which languages remain in each slice after filtering out mixed-language and unknown labels. This makes the role of the slices clearer: the Frisian slice is the target low-resource slice, the Dutch slice is a related higher-resource comparison point, and the random slice mainly contains high-resource languages such as English, German, Chinese and Russian.

Slice	**Evaluated label**	**Count**	**Share**
Frisian	`fry`	42	48.3%
Frisian	`eng`	20	23.0%
Frisian	`nld`	20	23.0%
Frisian	`deu`	3	3.4%
Frisian	`ces`	1	1.1%
Frisian	`zho`	1	1.1%
Dutch	`nld`	79	91.9%
Dutch	`eng`	6	7.0%
Dutch	`fra`	1	1.2%
Random	`eng`	32	35.6%
Random	`deu`	19	21.1%
Random	`zho`	16	17.8%
Random	`rus`	13	14.4%
Random	`jpn`	4	4.4%
Random	`spa`	3	3.3%
Random	`mlg`	1	1.1%
Random	`nld`	1	1.1%
Random	`ukr`	1	1.1%

Table 3.1: OWI Slice language composition after filtering out mixed-language and unknown labels.

3.3.3 Germanic Language Cluster

The second dataset was made for analysis of language similarity and is focused on the Germanic language cluster. It consists of Wikimedia language dumps in the languages Dutch, German, Frisian and English [24]. These languages were selected to keep the dataset centred around Frisian and to allow for an initial inspection of languages familiar to us. These dumps were extracted by using WikiExtractor [2] and turned into a single dataset using Resiliparse with a train, validation and test split. This split layout is the result of relying on Resiliparse's own tools to ensure compatibility with Resiliparse. The dataset itself is intended for evaluation rather than for any other purpose the split might suggest.

The purpose of the GLC dataset is different from the purpose of OWI Slice. OWI Slice measures behavior on manually corrected web data, while GLC creates a more controlled setting where language similarity between closely related Germanic languages can be examined more easily. This is particularly relevant for Frisian because a language identifier may produce plausible but wrong predictions among Dutch, German, English and Frisian even when the text is clean.

The source Wikipedia collections differed in size. We therefore balanced each split independently by downsampling every language to the smallest language count in that split. This produced 17,210 documents per language in the train split, 955 in validation, and 956 in test.

The split names are inherited from Resiliparse's dataset-generation format and do not represent a training procedure performed in this thesis. We use the balanced train split as the GLC evaluation corpus because it provides the largest fixed sample, containing 68,840 documents in total. The validation and test files remain part of the reproducible dataset layout but are not used for the reported GLC results.

3.3.4 CommonLID

CommonLID is a recently created dataset through the efforts of a large group of researchers with the explicit intent to create a web data evaluation benchmark. As currently very few such datasets exists that are publicly available. CommonLID is therefore included to broaden the web-data evaluation beyond the three OWI slices.

It is a web-oriented benchmark for language identification and therefore provides a useful comparison point for testing whether the patterns observed in OWI Slice also appear in a larger externally curated dataset [19]. The dataset consists of 109 languages, where the samples contain web data that was cleaned, curated and annotated per line. The cleaning and curating of the data as outlined in the accompanying paper [19] differs from the html extraction of Resiliparse. As a result, the dataset is a cleaner form of web-extracted data compared to OWI Slice.

3.4 Evaluated Language Identification Systems

After establishing the datasets, the next step is to define our systems that are evaluated on those datasets. The flow of the experiments is built around three components. Resiliparse is the system whose behavior we evaluate first, FastText is the stronger text-based fallback against which we compare it, and UrlExtractor is the metadata-based classifier and trigger source used in the web-specific adaptations. This section introduces those components in that order, because the later experiments follow the same progression from baseline, to fallback, to trigger-based and URL-based adaptation.

3.4.1 Resiliparse

Resiliparse as a whole is a collection of processing tools for parsing and analysing web data. Within this research we focus on its language-detection component, because that is the part of the pipeline that determines which language label is assigned to extracted web text. Resiliparse is therefore treated as the baseline system because it is the efficient first-pass component already used in this pipeline: the objective of this research is not to replace it, but evaluate how well the language identifier already used in this setting behaves for low-resource languages and web-extracted text.

We base our description on the documented Resiliparse interface and detector behaviour [7]. When we discuss adapted routing strategies or behaviour, we base our interpretation on our inspection of the implementation and the outputs observed during the experiments.

The relevant Resiliparse method is an implementation of a character n-gram profile approach. A document is turned into n-gram evidence and compared against language profiles, after which the languages are ranked by an out-of-place (OOP) score. This follows the same broad principle as the Cavnar-Trenkle method: the language whose profile is closest to the document receives the lowest score and is treated as the best candidate [6]. In the evaluated implementation, Resiliparse reports support for 101 languages through its language-detection profiles [7].

The ranking of a document against the provided profiles is important for the rest of the thesis because it gives more information than a single predicted label. With the right parameter Resiliparse can return all of the scores for the languages it was trained on. The rank-1 language is the final Resiliparse prediction, but the rank-2 and rank-3 languages, together with their OOP scores, show how clearly separated that prediction is from nearby alternatives. The later cutoff sweep, score-gap analysis, and language-distance-aware trigger all build on this point: the internal ranking can reveal uncertainty even when the final prediction looks like an ordinary single-label decision.

The default implementation of Resiliparse comes with a threshold value for the OOP score. If the threshold value is exceeded then Resiliparse returns unknown, unless the right parameters are provided that cause Resiliparse to pass the document to a FastText model.

3.4.2 FastText Model

FastText is the main alternative text-based model used in this research as it is Resiliparse's fallback. It is an efficient supervised text classifier that uses subword information, including character-level n-grams, to represent text [11, 10]. Compared with a fixed n-gram profile method, it requires labelled training data and a heavier model, but it can provide a stronger classification signal in cases where the statistical profile ranking is ambiguous.

In this thesis FastText has two roles. First, it is evaluated as an independent detector so that Resiliparse can be compared with a different language-identification approach. Second, it is used as the fallback model in the adapted systems. This second role is the more important one for the research design. As our goal is not simply to replace Resiliparse with FastText for every document. Web indexing requires high throughput, so the more relevant question is whether FastText can be reserved for the cases where Resiliparse is likely to be wrong. As FastText is expected to be slower than Resiliparse.

For that reason, the runtime comparison is part of the evaluation rather than an implementation detail. The adapted systems measure whether extra evidence can be used selectively. FastText can be reserved for cases where a cutoff, language-distance, or UrlExtractor trigger indicates that the faster Resiliparse result is fragile, while UrlExtractor can also be evaluated as a parallel fallback model.

3.4.3 UrlExtractor

The third source of evidence is UrlExtractor. Unlike Resiliparse and FastText, UrlExtractor does not classify the textual content of a document. It classifies from web-specific metadata that is available before or alongside text extraction. Prior work has shown that URLs can contain useful language clues, for example in domains, paths, or other URL features, but that URL-based classification is a different problem from content-based language identification [4]. This distinction is central to how UrlExtractor is used in this thesis as a way to gain additional information from outside the textual content.

UrlExtractor checks for language evidence in a fixed order. It first looks for language hints in the subdomain, for example (https://fy.ru.nl), where the fy is what we call the subdomain. Then for language-like path segments such as `/en/`, `/nl/`, or locale forms such as `/en-us/`. If these are not present, it checks country-code or language-associated top-level domains. If no explicit URL clue is found, it falls back to English. As most top-level domains that we cannot link to a language are most likely to be English. This final fallback is deliberately evaluated against an always-English baseline, because web data is often English-heavy and an English default can otherwise make the URL-based method look stronger than it really is.

In the implementation, the English fallback receives a low URL score and is treated differently from explicit URL evidence. A language found in the subdomain, path, or top-level domain is treated as an explicit URL clue. This matters because an absent URL signal should not be treated as equally strong evidence.

UrlExtractor is used in three ways. First, we evaluate it as a standalone metadata-based classifier, with the always-English baseline showing how much of its performance can be explained by the default English prediction. Second, we use the UrlExtractor prediction as the fallback label in RP+URL variants, parallel to the RP+FastText systems. Third, we use it inside the combined RP+FastText+URL systems. In the language-aware version, when Resiliparse is flagged as fragile, we use the URL language if explicit URL evidence supports either the rank-1 or rank-2 Resiliparse language. If it does not, we fall back to FastText. The split-trigger variant separates the two fallback causes: OOP-cutoff cases are routed to FastText, while rank-distance cases first use explicit UrlExtractor evidence and otherwise keep the Resiliparse prediction.

3.5 Baseline Evaluation

We perform the initial baseline evaluation with the WiLI-2018 dataset and the OWI slices. We run Resiliparse, FastText and the default Resiliparse+FastText combination. The default combination is a composite version designed for a balance between accuracy and efficiency, where Resiliparse handles the first pass and the instances where it lacks confidence are handled by FastText. In the baseline evaluation we ask whether Resiliparse performs differently on clean Wikipedia-style data than on heterogeneous web data, and whether this difference is especially visible for low-resource languages such as Frisian. The other datasets are also used and included within this section for comparison and reference but they are not the main focus.

We use the rsp module to extract from the datasets, run the LID systems and visualise the results. The results from running the datasets through the LID systems is stored for each dataset in `rp_outputs.csv` with the necessary data for reproducibility and to analyse and visualise. The visualisation is done through a number of figures aimed to analyze three aspects of the baseline. The first is a table that gives a model-by-dataset summary, meant to give us an overview of performance of each model and dataset.

Table 3.2: Dataset-level model performance across the main evaluation datasets.

Dataset	Model	Accuracy	F1 (macro)	F1 (weighted)	Support	Languages
CommonLID	FastText	0.681	0.388	0.629	373230	109
	RP+FastText	0.652	0.334	0.600	373230	109
	Resiliparse	0.620	0.297	0.571	373230	109
GLC	FastText	0.990	0.991	0.991	68840	4
	RP+FastText	0.970	0.982	0.982	68840	4
	Resiliparse	0.970	0.982	0.982	68840	4
OWI_slice_dutch	FastText	0.965	0.586	0.961	86	3
	RP+FastText	0.791	0.483	0.867	86	3
	Resiliparse	0.791	0.501	0.871	86	3
OWI_slice_frisian	FastText	0.954	0.792	0.950	87	6
	RP+FastText	0.471	0.270	0.428	87	6
	Resiliparse	0.471	0.270	0.428	87	6
OWI_slice_random	FastText	0.978	0.954	0.984	90	9
	RP+FastText	0.900	0.882	0.932	90	9
	Resiliparse	0.900	0.882	0.932	90	9
WiLI_Filtered	FastText	0.732	0.686	0.687	39000	77
	RP+FastText	0.672	0.610	0.611	39000	77
	Resiliparse	0.672	0.610	0.611	39000	77

The second set of figures are two bar graphs showing accuracy and macro-f1 score divided by clean data and web data.

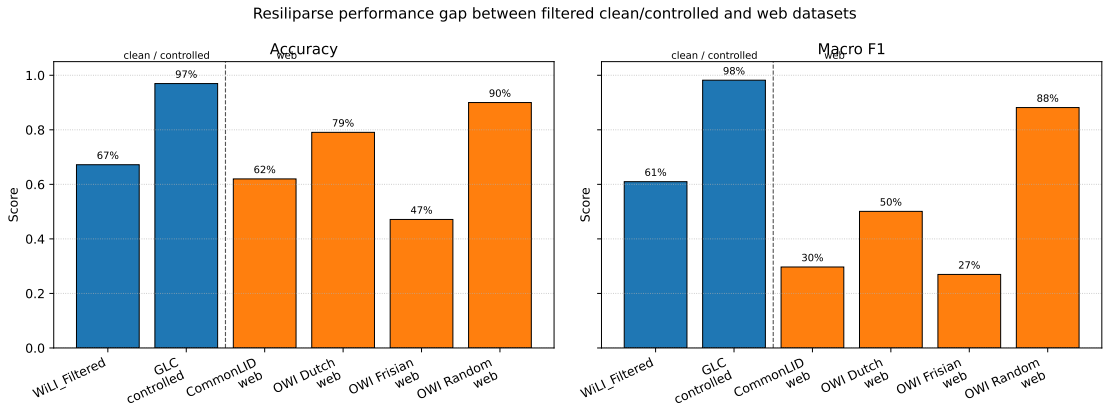

Figure 3.1: Resiliparse performance across clean, controlled, and web-domain datasets. The plot separates Wikipedia-style and controlled Germanic evaluation from CommonLID and OWI Slice web evaluation.

As defined in Section 2.1, we classify languages with fewer than 100,000 Wikipedia articles as low-resource, languages with at least 100,000 but fewer than one million articles as mid-resource, and languages with at least one million articles as high-resource. Figure 3.2 groups macro-F1 performance according to these categories. The one-million boundary separates the 19 largest Wikipedia editions from the remaining editions in the Wikimedia list consulted for this work [14].

The counts used in the analysis were recorded on 31 May 2026 and then kept fixed so that the generated comparisons remain reproducible. The Wikimedia list was accessed and archived later, on 13 June 2026. Since Wikipedia continues to change, the archived page contains slightly different article counts from those used to generate the figures. These differences did not change the assigned resource categories.

The implementation applies the fixed article-count lookup only to the union of language codes supported by the local Resiliparse and FastText evaluation pipeline. Supported languages without a recorded count, and dataset labels outside that set, are shown as unknown. For WiLI-2018, the grouping is applied to the filtered evaluation-overlap view so that the comparison is not dominated by labels that occur only in the broader WiLI benchmark.

Figure 3.2 examines whether the resource level of a language is reflected in model performance. If the amount of available language data is an important factor, we would expect performance to decrease from high-resource to mid-resource and then to low-resource languages. We use macro-F1 here because overall accuracy could hide poor performance on individual languages. The comparison to focus on is therefore the difference between the three resource groups within each dataset and model.

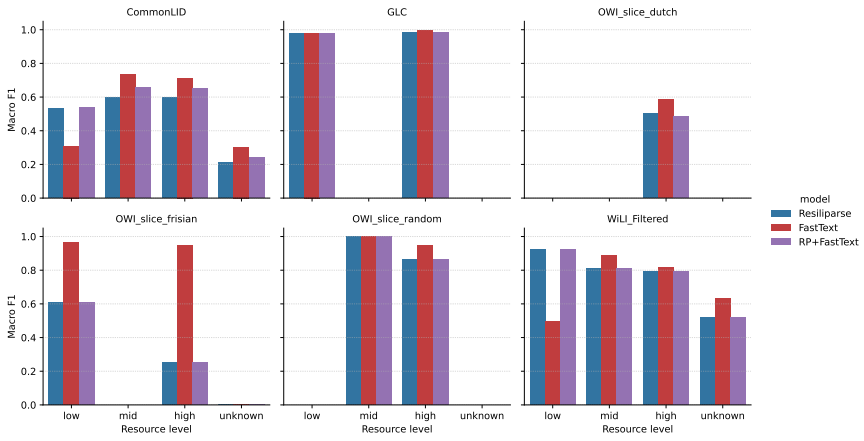

Figure 3.2: Macro-F1 grouped by operational language resource level. Low, mid, and high are defined by Wikipedia article-count thresholds within the local Resiliparse and FastText support lookup.

3.5.1 Discussion Baseline Evaluation

The baseline results quickly reveal that the problem is not simply that language identification is difficult in general. Resiliparse performs much better on the cleaner or controlled datasets than on the target Frisian OWI Slice, while FastText shows that many of the same documents remain classifiable by a stronger text-based model. The main gap is visible in the OWI Frisian row: Resiliparse reaches 47% accuracy and 0.270 macro-F1, while FastText reaches 95% accuracy and 0.792 macro-F1 on the same corrected slice.

Table 3.2 places this result in the broader baseline comparison. The Frisian OWI Slice performs poorly compared to the cleaner WiLI and GLC datasets. CommonLID is also web-domain data, but its scores are less severe than OWI Frisian, a consideration here is that CommonLID contains more languages and contains languages not supported by Resiliparse. And as the Dutch OWI Slice contains mainly high-resource language we can cautiously suggest that web noise is

likely another factor in performance. We also observe that in most cases Resiliparse and Resiliparse+FastText perform comparably. Given that Resiliparse calls FastText as a fallback based on the OOP score, this suggests that many of the misclassifications have an OOP score that is below the threshold value. The stronger performance of FastText suggests that better use of the fallback-mechanism or adjustment of the threshold value, may increase overall performance. In the following experiments we will attempt varying threshold values and attempt to find other factors and measures to help decide confidence for Resiliparse.

Figure 3.1 makes the deployment gap more explicit for Resiliparse. The difference in performance between the cleaner reference datasets and the web-domain datasets shows that the results of clean data cannot be simply applied to web-extracted data as well. We can safely conclude that there is a difference between the two types of datasets that sets them apart for LID. However, we do lack large datasets of web-extracted data that fulfill our requirements. This is a limitation within this research which should be considered for the results and any conclusions.

Figure 3.2 does not show one consistent relationship between resource level and performance. CommonLID most closely follows our expectation: all three systems perform worse for the low-resource group than for the high-resource group, with the largest difference visible for FastText. GLC shows only a small difference between its low- and high-resource groups.

The other datasets do not follow this pattern. In the Frisian OWI Slice, Resiliparse performs better for the low-resource group than for the high-resource group. The mid-resource group in OWI Random also performs better than its high-resource group, while `WiLI_Filtered` shows different patterns for FastText and the Resiliparse-based systems. The Dutch slice contains only a high-resource group and therefore provides no comparison.

We therefore do not find a consistent resource-level effect across the datasets. The figure suggests that resource level can be associated with lower performance, particularly in CommonLID, but it does not explain the results by itself. Our resource-level boundaries may need to be calibrated differently to reveal a clearer difference in performance. Languages classified as low-resource by our definition may still have substantially more digital resources than the languages most affected by data scarcity, while the high-resource category also combines languages with different levels of representation. The number and identity of languages within each group also affect these comparisons. The following sections therefore examine text length, linguistic similarity, cutoff behaviour, and domain mismatch as additional contributing factors.

3.6 Error Factor Analysis

To best determine the contributing factors on the performance of Resiliparse, we perform additional analysis of the earlier produced `rp_outputs.csv` as these files contain for each dataset row all the detector outputs and metadata collected from the dataset documents. We also perform an error analysis on the misclassifications of the Frisian OWI Slice by the annotators. Based on the literature and the preliminary analysis of the baseline, we investigate the impact of web noise, text-length, linguistic similarity, and the effects of Resiliparse's confidence.

3.6.1 Mixed Annotation Analysis

The mixed annotation analysis is performed on the list of pages annotated as containing more than one language by the annotators for the Frisian OWI Slice. It is a simple visual inspection of the texts which were annotated as mixed to determine what languages they actually were and if the mixed content was the result of web noise, such as boilerplate or other artifacts. Within the visual inspection we also have access to the classification of Resiliparse on these web pages. We also consider the disagreements between the annotators for both Frisian and random.

The analysis of the Frisian mixed annotations shows that combinations of linguistically similar languages are often present on web pages classified as Frisian. Along with this, some errors occur in pages or fragments where navigation menus makes up the additional language. Our further factor analysis therefore separates several possible causes rather than treating all web-data errors as one phenomenon. The majority of mixed annotations were made up of English, German, Dutch and partial Frisian.

3.6.2 Annotation Disagreements

The manual OWI labels also provide valuable information as an error analysis and a benchmark. The double-annotated Frisian slice has 96/100 raw agreement, with four manually resolved disagreements: id 47 was resolved as English, id 49 as Dutch, id 61 as Frisian, and id 62 as Mixed languages. The double-annotated random slice has 98/100 raw agreement, with id 6 resolved as Japanese and id 99 as Ukrainian. The stored disagreement records show the ambiguity behind these cases: English versus Unknown, Dutch versus Frisian, missing versus Frisian, missing versus Mixed languages, Japanese versus Chinese, and Russian versus Ukrainian. Dutch has only one annotator export, so no double-annotation agreement rate is reported for that slice. The high agreement supports the reliability of the corrected OWI labels, while the remaining disagreements show that web-language annotation can be difficult even for humans when pages are mixed, labels are missing, or languages are closely related or visually similar.

From these results we can already see that linguistic similarity likely has a part in the misclassifications by Resiliparse. As most of these annotations are made on originally indexed as Frisian webpages. However, we cannot draw any firm conclusions because the Frisian OWI Slice sample only contains 100 web pages. And the annotators cannot be considered language experts and they did also rely on external tools. Another consideration when drawing conclusions is that the annotators annotated boilerplate and navigation menu remnants as the language in which they were written. An annotation mistake caused by not clearly defining the annotation rules beforehand.

3.6.3 Text-Length

The text-length analysis examines whether Resiliparse's performance varies with the amount of available input text. Text length is measured in characters and grouped into length bins. These bins are chosen to roughly correspond to word-count milestones, while keeping the analysis itself character-based. The purpose of this sweep is to determine whether shorter web-extracted fragments play a major factor in the misclassification. Figure 3.3 shows the controlled Germanic Language Cluster setting, while Figure 3.4 shows the target Frisian OWI Slice.

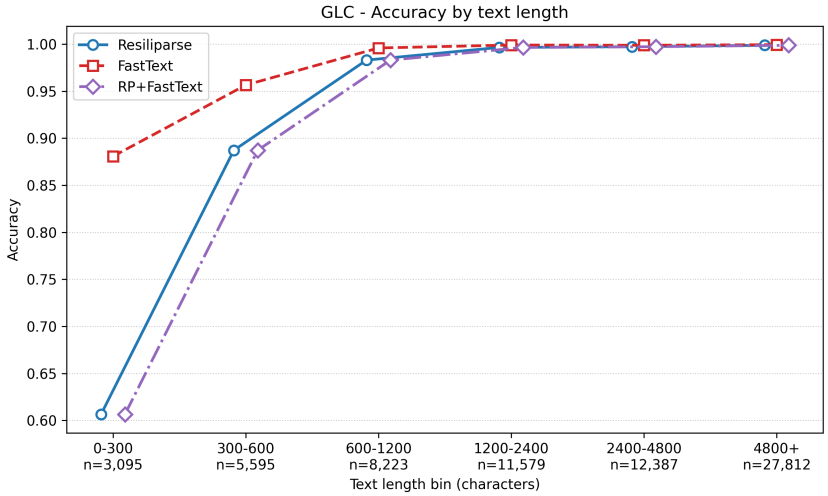

Figure 3.3: Accuracy by character-length bin for the Germanic Language Cluster. The bins approximate word-count milestones while keeping the analysis character-length based.

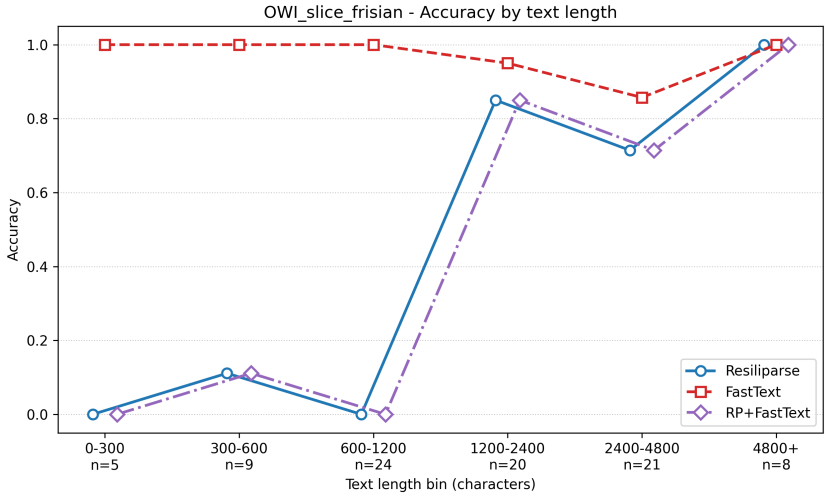

Figure 3.4: Accuracy by character-length bin for the Frisian OWI Slice. The number below each bin reports how many evaluated documents fall within that range.

3.6.4 Discussion Text-Length

The plots show that text length appears to impact accuracy significantly below 300 for GLC and below 1200 for Frisian OWI Slice with Resiliparse, with FastText appearing less impacted by text length. The literature broadly supports that text below a certain length is distinctly hard to classify. Our results appear to back that claim, although FastText does perform remarkably well compared to Resiliparse.

We also observe that accuracy decreases slightly in the 2400-4800 character bin, despite the

overall improvement with text length. No clear cause is immediately prevalent but the most likely cause is a difference in the documents within the bin. As the sample sizes are small any such deviations could be caused by the low sample size not averaging out the overall performance.

RP+FastText follows the same pattern as Resiliparse across the text-length bins, while FastText on its own performs considerably better. This suggests that the default OOP cutoff does not trigger FastText for many of the documents where it could improve the result. We cannot determine from the text-length figure alone whether this fully explains the overlap, but it does give us a reason to investigate the cutoff behaviour and the relationship between the highest-ranked Resiliparse candidates in the following experiments.

No strong conclusions can be drawn about Resiliparse or FastText from these results yet. As the use of web-extracted data makes it difficult to determine what other factors could be contributing. The GLC data performs significantly better, which could be attributed to being cleaner or closer to the training data than OWI data. The OWI data also shows a correlation between text length and model performance. However, it is too sparse for any conclusive statements. Overall, the results do support the existing literature on the impact of text length and we observe a clear correlation, which informs the further steps. The combination of GLC data along with OWI data both show a decrease in performance for the short text lengths as expected. Despite the limitations of our results, we can cautiously conclude that text length is one of the factors present. However, it does not account for all errors observed. Therefore, in the next experiments we will attempt to isolate some of the other potential factors.

3.6.5 Linguistic Similarity

The classification of multiple languages is already a challenging task. We suspect that distinguishing between closely related languages adds a further layer of difficulty. As Resiliparse relies on language profiles constructed from character n-grams, languages that share vocabulary and similar frequency distributions may produce similar profile rankings. If this is the case, we would expect the correct language to remain near the top of the ranking even when the final prediction is incorrect.

To investigate this, the GLC experiment examines not only Resiliparse's top prediction but also the second- and third-ranked language candidates. When the first-ranked language is incorrect, we analyse whether the correct language appears in these lower ranks and how large the OOP score difference is between the competing candidates. Small OOP score gaps would indicate that Resiliparse considered the alternatives similarly plausible.

The GLC dataset was specifically constructed to isolate the effect of linguistic similarity. By focusing on Frisian and its neighbouring Germanic languages, the experiment removes much of the web-extraction noise present in the OWI data and allows the impact of language similarity to be examined more directly. The underlying assumption is that Resiliparse is often sufficiently confident that a fallback system such as FastText is not triggered, while still struggling to separate closely related languages accurately.

The same language-similarity analysis is also applied to the OWI slices, CommonLID and the other datasets. This links the controlled GLC evidence back to web data and checks whether the ambiguity patterns are visible in the target setting as well. Figure 3.5 shows where Resiliparse sends Germanic GLC documents when it is wrong.

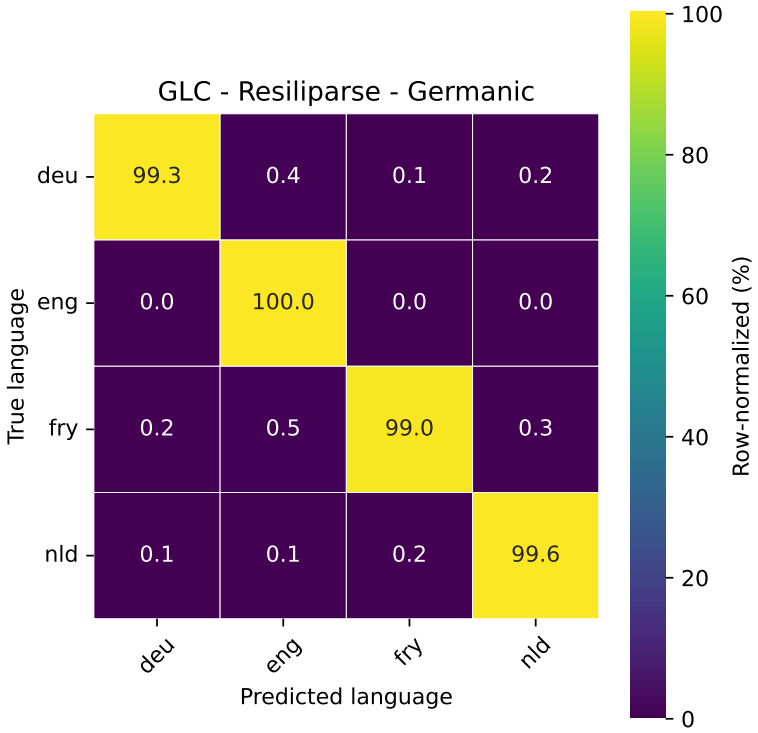

Figure 3.5: Germanic confusion matrix for Resiliparse on the GLC dataset.

Table 3.3 then divides this confusion pattern to the internal rank separation. So, we can see how much the scores differ and how often they are right.

Table 3.3: Rank-1/rank-2 OOP score gap by correctness category for GLC and the OWI web slices. Correctness records whether the true language appears at rank 1, rank 2, rank 3, or none of these reported ranks. Lower gaps indicate that Resiliparse placed the first two language candidates closer together.

Dataset	Rank correctness	N	Median gap	% gap ≤ 10	% gap ≤ 25	% gap ≤ 50	% gap ≤ 100
CommonLID	Rank 1 correct	231376	70.0	5.5%	15.1%	34.4%	68.9%
	Rank 2 correct	13641	8.0	58.7%	89.1%	98.7%	100.0%
	Rank 3 correct	6093	9.0	56.7%	88.7%	98.8%	100.0%
	Neither rank correct	122120	31.0	26.2%	45.6%	61.5%	78.3%
GLC	Rank 1 correct	66746	78.0	2.3%	7.2%	21.0%	75.1%
	Rank 2 correct	848	5.0	77.0%	98.7%	100.0%	100.0%
	Rank 3 correct	357	6.0	77.0%	97.8%	100.0%	100.0%
	Neither rank correct	889	6.0	73.7%	95.7%	99.7%	100.0%
OWI_slice_dutch	Rank 1 correct	68	39.0	10.3%	32.4%	54.4%	98.5%
	Rank 2 correct	7	8.0	57.1%	100.0%	100.0%	100.0%
	Rank 3 correct	3	6.0	66.7%	100.0%	100.0%	100.0%
	Neither rank correct	8	5.0	75.0%	87.5%	100.0%	100.0%
OWI_slice_frisian	Rank 1 correct	41	31.0	14.6%	31.7%	70.7%	97.6%
	Rank 2 correct	18	6.0	83.3%	100.0%	100.0%	100.0%
	Rank 3 correct	7	5.0	71.4%	100.0%	100.0%	100.0%
	Neither rank correct	21	6.0	61.9%	95.2%	100.0%	100.0%
OWI_slice_random	Rank 1 correct	81	60.0	2.5%	12.3%	39.5%	84.0%
	Rank 2 correct	7	8.0	71.4%	100.0%	100.0%	100.0%
	Rank 3 correct	1	24.0	0.0%	100.0%	100.0%	100.0%
	Neither rank correct	1	2.0	100.0%	100.0%	100.0%	100.0%
WiLI_Filtered	Rank 1 correct	26206	69.0	3.9%	13.1%	34.6%	66.7%
	Rank 2 correct	676	6.0	67.8%	91.9%	98.8%	100.0%
	Rank 3 correct	203	8.0	57.1%	86.2%	96.6%	100.0%
	Neither rank correct	11915	14.0	42.4%	68.9%	86.4%	97.5%

3.6.6 Discussion Linguistic Similarity

The results of Figure 3.5 show us that for GLC, the clean data, we have a few mistakes that can be explained by linguistic similarity but not any significant amount. However, Table 3.3 provides evidence that linguistic similarity contributes to the observed errors. In CommonLID, when the correct language appears at rank 2 or rank 3, the median OOP gap falls from 70.0 for correct rank-1 predictions to 8.0 and 9.0 respectively. Furthermore, 89.1% of rank-2 cases and 88.7% of rank-3 cases have a gap of 25 or less. This indicates that a significant number of misclassifications occur where the rank-2 or rank-3 candidate is the correct language label.

The same pattern is visible in the OWI slices. For the Frisian OWI Slice, the median gap decreases from 31.0 for correct rank-1 predictions to 6.0 and 5.0 when the correct language appears at rank 2 or rank 3. Likewise, all rank-2 and rank-3 cases fall within a gap of 25 or less. Similar behaviour is observed for the OWI Dutch and OWI Random slices, where the correct language frequently remains among the highest-ranked candidates despite not being selected as the final prediction. The pattern is much weaker when we look at GLC and `WiLI_Filtered`, which are clean data.

Taken together, these results suggest that a portion of the observed errors are associated with weak separation between competing language candidates rather than the complete absence of the correct language from the ranking. The difference in the results between the clean and web-extracted data here suggests that a level of web noise is in part responsible for the weaker separation between these languages. This pattern suggests that a combination of factors would be responsible for the decreased performance of low-resource languages on web-extracted data.

We also observe that the effect is weaker for the OWI Dutch and OWI Random slices than for CommonLID and the Frisian OWI Slice. This is expected, as the Dutch and Random slices contain predominantly high-resource languages, which we know from Table 3.1, for which language profiles are generally more distinct and therefore less likely to produce ambiguity between closely related candidates.

Another limitation is the relatively small sample size of the OWI slices. Several rank-2 and rank-3 categories contain only a limited number of observations, making the reported percentages sensitive to individual documents. We therefore use CommonLID as a larger reference dataset to support the analysis.

Finally, the OOP gap is used as a measure of ranking ambiguity rather than linguistic similarity itself. A small gap indicates that Resiliparse assigned similar scores to multiple candidate languages, but does not by itself explain why those scores are similar. Linguistic similarity is one possible explanation, but other factors such as text length or web-extraction noise may also contribute.

3.6.7 Cutoff Sweep

The cutoff sweep experiment tests a architectural part of Resiliparse's routing behavior. As we previously mentioned, Resiliparse uses OOP scores in its n-gram ranking of language profiles and can route uncertain cases to FastText. The default cutoff is designed to catch unreliable statistical predictions, often caused by very short or noisy texts. What we aim to verify here is how good the existing default value of 1200 is at catching wrong classifications. We will run the cutoff sweep on Resiliparse, FastText, Resiliparse+FastText and a version of Resiliparse+FastText with slightly different routing behaviour. Resiliparse routes documents its unsure of to FastText already, the added behaviour here is that if Resiliparse's rank-1 and rank-2 language are linguistically close, it also routes to FastText. This is already an adapted version included early in an attempt to verify if the direction of linguistic similarity for improvement holds any ground. It makes use of `langcodes tag_distance` a Python library that returns a distance score based on ISO3 language codes.

Table 3.4: `langcodes` tag distance values and their interpretation. Adapted from the `langcodes` documentation on PyPI.

Value	Meaning	Example
0	These codes represent the same language, possibly after filling in values and normalizing.	Norwegian Bokmål → Norwegian
1–3	These codes indicate a minor regional difference.	Australian English → British English
4–9	These codes indicate a significant but unproblematic regional difference.	American English → British English
10–24	A gray area that depends on the use case. There may be problems with understanding or usability.	Afrikaans → Dutch; Wu Chinese → Mandarin Chinese
25–50	These languages are not similar, but there are demographic reasons to expect some intelligibility.	Tamil → English; Marathi → Hindi
51–79	There are large barriers to understanding.	Japanese → Japanese in Hepburn romanization
80–99	These are different languages written in the same script.	English → French; Arabic → Urdu
100+	These languages have nothing particularly in common.	English → Japanese; English → Tamil

We perform a sweep of values for the cutoff point over the detector outputs we generated before. This experiment along with the earlier linguistic similarity will be the basis for determining the way in which we adapt Resiliparse.

Figure **??** links the observed ambiguity to the cutoff mechanism. The cutoff sweep tests whether the default OOP threshold catches fragile Germanic cases or leaves them inside the statistical Resiliparse path.

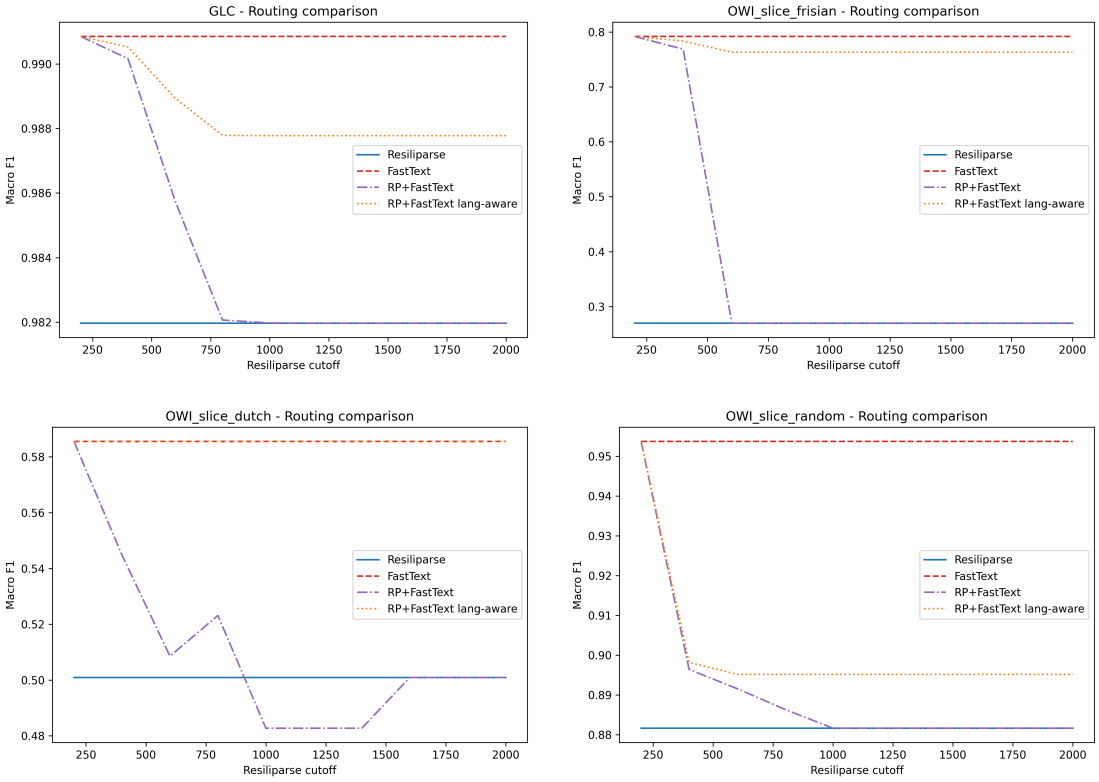

Figure 3.6: Cutoff sweeps for GLC and the three OWI slices, showing Resiliparse, FastText, standard RP+FastText, and language-aware RP+FastText across the evaluated cutoff values.

3.6.8 Discussion Cutoff Sweep

We observe that FastText achieves the highest macro-F1 score in both the GLC and OWI Frisian settings and remains unaffected by the routing threshold. This is consistent with the baseline evaluation, where FastText consistently outperformed Resiliparse on both datasets.

For both datasets, the routing approaches follow the expected pattern. At lower cutoff values, performance remains close to the FastText baseline because more documents are forwarded to FastText. As the cutoff increases, fewer documents are routed and performance gradually converges towards the Resiliparse baseline. In the GLC setting, the standard RP+FastText configuration stabilises at approximately the 1200 cutoff at a macro-f1 slightly below 0.982, which corresponds to the default threshold used by Resiliparse. Beyond this point, further increases in the cutoff have little effect on the overall result. In the Frisian OWI Slice, we see that performance bottoms out around 600 at a macro-f1 below 0.3, which is understandable as the baseline evaluation put accuracy at 47%.

The language-aware routing variant consistently outperforms the standard RP+FastText configuration on the GLC dataset, indicating that language-specific routing information can recover errors that are not addressed by the threshold-based approach alone. The same effect is visible on the Frisian OWI Slice, where the language-aware routing stays far closer to FastText than the Resiliparse line across the evaluated cutoff range. This suggests that the additional language-aware logic is beneficial.

These results show that language-aware routing can improve performance over Resiliparse and the standard cutoff-based fallback. Whether the increased use of FastText outweighs the added runtime is evaluated later in Section 3.7. So, while the language-aware variant shows promise, its practical value within the OWI setting is dependent on the fallback model's efficiency and accuracy.

3.6.9 Discussion Training Data and Domain Mismatch

The previous analyses do not fully explain every misclassification. Some remaining errors likely reflect the limitation from the training data, which was derived from Wikipedia. The mismatch between the clean data and web-extracted data, along with the domain of Wikipedia differing from web-extracted data. The domain mismatch here refers not to topics. Wikipedia covers many topics but its style and structure are very consistent internally and still differ from the rest of the web.

For this reason, we do consider domain mismatch as a residual factor. WiLI-2018 and GLC represent cleaner Wikipedia-derived text, while OWI Slice and CommonLID represent web data. If the same low-resource language performs well on Wikipedia-style text and poorly on OWI web text, then the gap can plausibly involve both extraction noise and domain shift. However, when the confusion and ranking analyses point specifically to nearby Germanic languages, language similarity becomes a factor which likely has a larger impact on the Frisian misclassifications.

3.7 Adapted LID Approaches

The adapted-system experiments evaluate whether the factors identified in the error analysis can be targeted without abandoning the efficiency of Resiliparse's design. Resiliparse is useful in web indexing because its statistical language identifier is fast and broad, allowing for easier language set expansion, while FastText is more expensive and has a more limited language set. The adaptation therefore does not replace Resiliparse with FastText globally or replace the fast first pass wholesale. Instead, it tests routing rules that send only selected documents to FastText or UrlExtractor-supported fallback decisions.

3.7.1 URL/Domain-Based Adaptation

We use UrlExtractor both as a classifier and as part of a combined fallback decision. The motivation for this is that web documents often contain language information outside the extracted text itself. Domain names, URL paths, and country-code top-level domains can contain language cues that are not captured by a text-only language identifier. This information is not reliable enough to replace content-based LID in general, but it may still provide useful evidence when the text-based prediction is fragile without increasing runtime substantially.

Table 3.5 separates UrlExtractor from the detector systems on the Frisian OWI Slice, where URL metadata is available. The Always English row is included because the English fallback can otherwise make URL-derived evidence look stronger than it is.

Table 3.5: UrlExtractor and detector comparison on the Frisian OWI Slice.

Dataset	Model	Accuracy	F1 (macro)	F1 (weighted)	Support	Mean runtime (ms)	Docs/s
OWI_slice_frisian	Always English	0.230	0.062	0.086	87	0.000	n/a
	URL	0.954	0.787	0.949	87	0.032	30895.2
	Resiliparse	0.471	0.270	0.428	87	0.198	5052.6
	FastText	0.954	0.792	0.950	87	0.497	2012.2
	RP+URL	0.471	0.270	0.428	87	0.198	5052.6
	RP+URL lang-aware	0.862	0.765	0.866	87	0.227	4412.6
	RP+FastText+URL lang-aware	0.874	0.775	0.879	87	0.354	2826.2
	RP+FastText+URL split-trigger	0.782	0.719	0.784	87	0.223	4486.9

UrlExtractor performs strongly on the Frisian OWI Slice, with the Always English row to keep it in context. To understand the remarkably high UrlExtractor result on the Frisian OWI Slice, we visually inspected the URLs and their extracted language clues. Of the 87 evaluated documents, 75 contain an explicit language clue in the subdomain, path, or top-level domain. For 73 of these documents, the extracted URL language matches the annotated language.

We compare this result with the other two OWI slices in Table 3.6.

Table 3.6: UrlExtractor and detector comparison on the Dutch and random OWI slices.

Dataset	Model	Accuracy	F1 (macro)	F1 (weighted)	Support	Mean runtime (ms)	Docs/s
OWI_slice_dutch	Always English	0.070	0.043	0.009	86	0.000	n/a
	URL	0.779	0.394	0.825	86	0.054	18682.2
	Resiliparse	0.791	0.501	0.871	86	0.243	4118.4
	FastText	0.965	0.586	0.961	86	3.686	271.3
	RP+URL	0.791	0.483	0.867	86	0.243	4110.9
	RP+URL lang-aware	0.779	0.394	0.825	86	0.296	3374.5
	RP+FastText+URL lang-aware	0.965	0.586	0.961	86	0.464	2156.4
	RP+FastText+URL split-trigger	0.930	0.525	0.937	86	0.288	3474.7
OWI_slice_random	Always English	0.356	0.058	0.187	90	0.000	n/a
	URL	0.678	0.679	0.678	90	0.036	27976.3
	Resiliparse	0.900	0.882	0.932	90	0.245	4082.6
	FastText	0.978	0.954	0.984	90	0.827	1209.2
	RP+URL	0.900	0.882	0.932	90	0.245	4082.6
	RP+URL lang-aware	0.678	0.728	0.689	90	0.279	3586.0
	RP+FastText+URL lang-aware	0.944	0.895	0.962	90	0.706	1417.2
	RP+FastText+URL split-trigger	0.822	0.844	0.879	90	0.262	3821.0

3.7.2 Discussion URL/Domain-Based Adaptation

The results show that metadata can provide a useful additional signal in the OWI setting. This is particularly relevant for web-extracted data, where language evidence may exist outside the extracted text itself. Although UrlExtractor was originally considered as an auxiliary support for Resiliparse, it achieves the strongest results on the evaluated OWI slices. While this suggests that URL-based language identification may have potential as a standalone approach, our visual inspection indicates that the observed performance is based on clear language clues, with 73 of the 75 URLs containing such clues matching the annotated language. However, as we will discuss next this is not standard among web-extracted data.

UrlExtractor reaches 0.779 accuracy on the Dutch slice and 0.678 on the random slice, compared with 0.954 on the Frisian slice. This variation suggests that performance is strongly influenced by dataset composition and the availability of explicit language clues within the URLs. We therefore do not interpret the Frisian result as evidence that URL-based language identification performs equally well on web data in general.

Instead, we view these results as evidence that metadata can provide valuable support when content-based predictions become unreliable. This is particularly relevant in OWI-style data,

where language information may be present in domains, subdomains, or URL paths even when the extracted text itself provides a weak signal.

A second limitation is that URL evidence is unreliable. Some documents contain clear language clues, while others contain no useful metadata and fall back to a default prediction. The English default is particularly important here, because it acts like a most-popular-language baseline on datasets where English is common and can therefore make the method appear stronger than it is. Conversely, it becomes a disadvantage in datasets containing substantially less English. The always-English baseline therefore remains an important reference point when interpreting the UrlExtractor results.

We also note that the OWI slices are relatively small, meaning that the observed performance may be sensitive to data sparsity and dataset composition. A larger web-extracted benchmark such as CommonLID would provide a useful comparison, but URL metadata is not available. We therefore cannot draw strong conclusions about how well the approach generalises beyond the evaluated slices. Instead, we conclude that URL metadata is a valuable supplementary signal in OWI-style data and may be useful as either a lightweight first-pass prediction or an additional confidence signal within a composite system.

3.7.3 Language-Aware FastText Trigger

The language-aware trigger uses the fact that Resiliparse returns its top ranked languages together with the corresponding OOP scores. Instead of only looking at the first-ranked prediction, the adapted system also checks the relationship between the top candidates. The key condition is whether the rank-1 and rank-2 languages are close enough that the rank-1 prediction should be treated as fragile.

The adaptation uses two trigger mechanisms derived from the Resiliparse output. The first is the default OOP cutoff. The second is a language-aware trigger that treats a prediction as fragile when the rank-1 and rank-2 candidates are sufficiently close according to the `langcodes` tag distance. The RP+URL and RP+FastText variants use only the default OOP cutoff. The language-aware variants extend this by also triggering fallback on the rank-distance condition.

For the combined RP+FastText+URL language-aware system, UrlExtractor is consulted first when a document is flagged as fragile. If UrlExtractor supports the Resiliparse ranking, its prediction is used. Otherwise the system falls back to FastText. The RP+FastText+URL split-trigger variant separates the two trigger conditions. OOP-cutoff cases are routed to FastText, while rank-distance cases are routed to UrlExtractor. If UrlExtractor does not provide useful support, the original Resiliparse prediction is retained.

The adapted models are evaluated as an additive component analysis. The baseline is Resiliparse, with FastText and UrlExtractor also evaluated as independent classifiers. The fallback systems then split into simple and language-aware variants. This ladder is necessary because the evaluation is not only concerned with whether the final system is better, but also with which source of evidence contributes to the improvement.

Figure 3.7 first shows the OWI Frisian progression across the adaptation ladder. We use this figure as the visual entry point because it makes the target-slice progression easier to read before the numerical table.

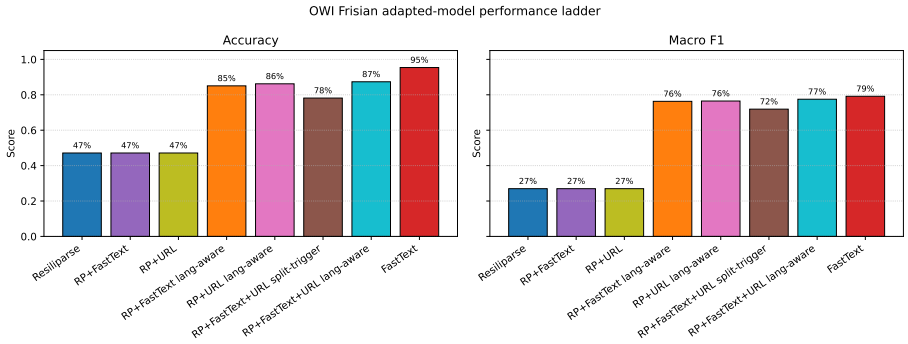

Figure 3.7: OWI Frisian adapted-model performance ladder. The figure shows the stepwise movement from Resiliparse to cutoff fallback, language-aware fallback, UrlExtractor-supported routing, and FastText.

The ladder shows that the cutoff-only systems remain close to the Resiliparse baseline, while the language-aware and combined systems move much closer to FastText. This suggests that the fallback model alone is not sufficient. The trigger must also identify rows where fallback is beneficial.

Table 3.7: Adapted-system performance on the OWI Frisian target slice.

Dataset	Model	Accuracy	F1 (macro)	F1 (weighted)	Support	Languages
OWI_slice_frisian	Resiliparse	0.471	0.270	0.428	87	6
	FastText	0.954	0.792	0.950	87	6
	RP+FastText	0.471	0.270	0.428	87	6
	RP+URL	0.471	0.270	0.428	87	6
	RP+FastText lang-aware	0.851	0.763	0.856	87	6
	RP+URL lang-aware	0.862	0.765	0.866	87	6
	RP+FastText+URL lang-aware	0.874	0.775	0.879	87	6
	RP+FastText+URL split-trigger	0.782	0.719	0.784	87	6

Table 3.7 gives the same target-slice result numerically. Resiliparse reaches 47% accuracy on OWI Frisian, while FastText reaches 95%. The cutoff-only RP+FastText and RP+URL systems remain at 47%. The rank-distance trigger improves RP+FastText lang-aware to 85%, and the combined RP+FastText+URL lang-aware row reaches 87%.

3.7.4 Discussion Language-Aware FastText Trigger

The results suggest that the language-aware trigger is substantially more effective than the default OOP cutoff on the Frisian OWI Slice. While RP+URL remains identical to Resiliparse at 47.1% accuracy, adding the rank-distance trigger raises RP+URL lang-aware to 86.2%. The unchanged performance of the cutoff-only system suggests that the default OOP threshold does not select many of the rows that were identified as problematic in the earlier error analysis. A similar pattern is visible for the combined approach, where RP+FastText+URL lang-aware reaches 87.4%. This indicates that many of the problematic cases identified in the earlier error analysis are not captured by the default cutoff alone. And that the introduced language-aware triger does find some of these through the relationship between the top-ranked Resiliparse candidates.

The split-trigger result provides additional context. Although it improves substantially over the Resiliparse baseline, it remains below the fully language-aware variants. This suggests that

the OOP cutoff and rank-distance conditions identify different subsets of documents, and that combining these signals is more effective than treating them independently. We observe here that the trigger allows for improvements in performance, these improvements are mostly the result of selectively using FastText more. Runtime is also still important, because the goal is not simply to optimise for performance but also to maintain high throughput.

The important limitation is that these trigger conditions are indirect. A small OOP gap or a close rank-1/rank-2 relationship indicates ambiguity in the Resiliparse ranking, but it does not prove why that ambiguity exists. Linguistic similarity is one possible explanation, especially for Frisian and nearby Germanic languages, but other factors such as short text length, extraction noise, or weak language profiles may produce similar score patterns.

The language-aware trigger should therefore be interpreted as a practical heuristic rather than a direct measurement of linguistic similarity. Its value depends on whether the ranking behaviour helps identify rows where Resiliparse is likely to be wrong.

3.7.5 Comparative Performance and Runtime

We compare the adapted approaches on the same datasets and with the same metrics as the baseline. The main target remains the Frisian OWI Slice, because this is where the low-resource web-data failure was most visible. OWI Dutch, OWI Random, GLC, `WiLI_Filtered`, and CommonLID are used as control settings so that the adaptation is not judged only on a single slice.

After the OWI Frisian results, the remaining comparison checks whether the same pattern appears outside the target slice, whether the adapted model changes the Germanic error pattern, and how much runtime cost is introduced by the adapted systems.

Table 3.8 shows the adapted-system results on the control datasets. On CommonLID, RP+FastText lang-aware reaches 0.677 accuracy and 0.382 macro-F1, close to FastText at 0.681 accuracy and 0.388 macro-F1. On `WiLI_Filtered`, RP+FastText lang-aware reaches 0.726 accuracy and 0.680 macro-F1, again close to FastText at 0.732 accuracy and 0.686 macro-F1. On GLC, RP+FastText lang-aware improves over Resiliparse from 0.970 to 0.983 accuracy, while FastText remains highest at 0.990 accuracy. The OWI Dutch slice shows RP+FastText lang-aware and RP+FastText+URL lang-aware matching FastText at 0.965 accuracy. On OWI Random, the language-aware variants improve over Resiliparse but remain below FastText.

Table 3.8: Adapted-system performance on control datasets.

Dataset	Model	Accuracy	F1 (macro)	F1 (weighted)	Support	Languages
CommonLID	Resiliparse	0.620	0.297	0.571	373230	109
	FastText	0.681	0.388	0.629	373230	109
	RP+FastText	0.652	0.334	0.600	373230	109
	RP+FastText lang-aware	0.677	0.382	0.626	373230	109
GLC	Resiliparse	0.970	0.982	0.982	68840	4
	FastText	0.990	0.991	0.991	68840	4
	RP+FastText	0.970	0.982	0.982	68840	4
	RP+FastText lang-aware	0.983	0.988	0.988	68840	4
OWI_slice_frisian	Resiliparse	0.471	0.270	0.428	87	6
	FastText	0.954	0.792	0.950	87	6
	RP+FastText	0.471	0.270	0.428	87	6
	RP+URL	0.471	0.270	0.428	87	6
	RP+FastText lang-aware	0.851	0.763	0.856	87	6
	RP+URL lang-aware	0.862	0.765	0.866	87	6
	RP+FastText+URL lang-aware	0.874	0.775	0.879	87	6
	RP+FastText+URL split-trigger	0.782	0.719	0.784	87	6
OWI_slice_dutch	Resiliparse	0.791	0.501	0.871	86	3
	FastText	0.965	0.586	0.961	86	3
	RP+FastText	0.791	0.483	0.867	86	3
	RP+URL	0.791	0.483	0.867	86	3
	RP+FastText lang-aware	0.965	0.586	0.961	86	3
	RP+URL lang-aware	0.779	0.394	0.825	86	3
	RP+FastText+URL lang-aware	0.965	0.586	0.961	86	3
	RP+FastText+URL split-trigger	0.930	0.525	0.937	86	3
OWI_slice_random	Resiliparse	0.900	0.882	0.932	90	9
	FastText	0.978	0.954	0.984	90	9
	RP+FastText	0.900	0.882	0.932	90	9
	RP+URL	0.900	0.882	0.932	90	9
	RP+FastText lang-aware	0.944	0.895	0.962	90	9
	RP+URL lang-aware	0.678	0.728	0.689	90	9
	RP+FastText+URL lang-aware	0.944	0.895	0.962	90	9
	RP+FastText+URL split-trigger	0.822	0.844	0.879	90	9
WiLI_Filtered	Resiliparse	0.672	0.610	0.611	39000	77
	FastText	0.732	0.686	0.687	39000	77
	RP+FastText	0.672	0.610	0.611	39000	77
	RP+FastText lang-aware	0.726	0.680	0.681	39000	77

Figure 3.8 compares the baseline and the full adapted model on the OWI Frisian Germanic subset. In the Resiliparse matrix, 85.0% of English rows and 85.0% of Dutch rows are assigned to Frisian. In the adapted matrix, English reaches 100.0% correct and Dutch reaches 65.0% correct. The Frisian row remains almost unchanged, moving from 97.1% to 97.5% correct.

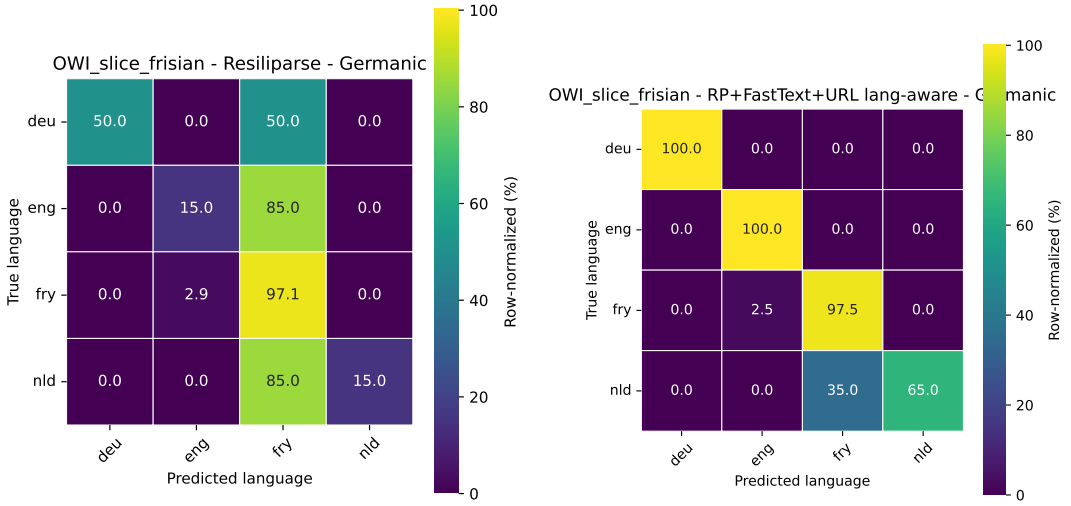

Figure 3.8: Germanic confusion matrices for OWI Frisian, comparing baseline Resiliparse with the combined RP+FastText+URL language-aware adaptation.

Table 3.9 reports speedup relative to Resiliparse and FastText. On OWI Frisian, Resiliparse processes 5052.6 documents per second and FastText processes 2012.2 documents per second. The full RP+FastText+URL lang-aware system processes 2826.2 documents per second, while the split-trigger variant processes 4486.9 documents per second. On OWI Dutch and OWI Random, the full language-aware system remains faster than FastText but slower than Resiliparse. RP+FastText lang-aware is slower than FastText on OWI Frisian and GLC.

Table 3.9: Runtime speedup comparison between detector systems using Resiliparse and FastText as baselines.

Dataset	Model	Macro F1	Mean (ms)	Docs/s	Speedup vs Resiliparse	Speedup vs FastText
CommonLID	Resiliparse	0.297	0.116	8605.1	1.00x	0.83x
	FastText	0.388	0.096	10404.7	**1.21x**	**1.00x**
	RP+FastText	0.334	0.121	8248.5	0.96x	0.79x
	RP+FastText lang-aware	0.382	0.211	4742.2	0.55x	0.46x
GLC	Resiliparse	0.982	0.367	2725.3	**1.00x**	**4.62x**
	FastText	0.991	1.695	589.9	0.22x	1.00x
	RP+FastText	0.982	0.367	2725.2	1.00x	4.62x
	RP+FastText lang-aware	0.988	2.027	493.4	0.18x	0.84x
OWI_slice_dutch	Resiliparse	0.501	0.243	4118.4	**1.00x**	**15.18x**
	FastText	0.586	3.686	271.3	0.07x	1.00x
	URL	0.394	0.054	18682.2	4.54x	68.87x
	RP+FastText	0.483	0.243	4108.1	1.00x	15.14x
	RP+URL	0.483	0.243	4110.9	1.00x	15.15x
	RP+FastText lang-aware	0.586	3.929	254.5	0.06x	0.94x
	RP+URL lang-aware	0.394	0.296	3374.5	0.82x	12.44x
	RP+FastText+URL split-trigger	0.525	0.288	3474.7	0.84x	12.81x
	RP+FastText+URL lang-aware	0.586	0.464	2156.4	0.52x	7.95x
OWI_slice_frisian	Resiliparse	0.270	0.198	5052.6	**1.00x**	**2.51x**
	FastText	0.792	0.497	2012.2	0.40x	1.00x
	URL	0.954	0.032	30895.2	6.11x	15.35x
	RP+FastText	0.270	0.198	5052.6	**1.00x**	**2.51x**
	RP+URL	0.270	0.198	5052.6	**1.00x**	**2.51x**
	RP+FastText lang-aware	0.763	0.642	1558.0	0.31x	0.77x
	RP+URL lang-aware	0.765	0.227	4412.6	0.87x	2.19x
	RP+FastText+URL split-trigger	0.719	0.223	4486.9	0.89x	2.23x
	RP+FastText+URL lang-aware	0.775	0.354	2826.2	0.56x	1.40x
OWI_slice_random	Resiliparse	0.882	0.245	4082.6	**1.00x**	**3.38x**
	FastText	0.954	0.827	1209.2	0.30x	1.00x
	URL	0.679	0.036	27976.3	6.85x	23.14x
	RP+FastText	0.882	0.245	4082.6	**1.00x**	**3.38x**
	RP+URL	0.882	0.245	4082.6	**1.00x**	**3.38x**
	RP+FastText lang-aware	0.895	1.029	971.6	0.24x	0.80x
	RP+URL lang-aware	0.728	0.279	3586.0	0.88x	2.97x
	RP+FastText+URL split-trigger	0.844	0.262	3821.0	0.94x	3.16x
	RP+FastText+URL lang-aware	0.895	0.706	1417.2	0.35x	1.17x
WiLI_Filtered	Resiliparse	0.610	0.140	7153.1	1.00x	0.96x
	FastText	0.686	0.134	7435.5	**1.04x**	**1.00x**
	RP+FastText	0.610	0.140	7147.7	1.00x	0.96x
	RP+FastText lang-aware	0.680	0.265	3780.4	0.53x	0.51x

3.7.6 Discussion Comparative Performance and Runtime

The adapted-system results show that the largest improvements on OWI Frisian come from the language-aware and combined systems rather than from the default cutoff alone. This matches the earlier error analysis, where many of the misclassifications were not simply low-confidence predictions according to the default OOP threshold. The factors involved combine in a way that degrades overall performance. The combination of cutoff with the rank-distance trigger manage to improve the degraded performance again. However, the fallback classifier can only help if the trigger selects the rows where Resiliparse is wrong.

The control datasets help us see that these adapted systems are not more effective for all settings. The strongest positive result remains the Frisian OWI Slice, which naturally is to be expected as it contains the most challenges to LID. However, this conclusion is also limited by the small size of the OWI slices, where a small number of documents can noticeably affect the reported percentages.

The confusion comparison shows that the adapted model shows improvement for the misclassifications into related languages, especially by reducing the number of English and Dutch

rows assigned to Frisian. However, the component contributions are not fully separable. The language-aware trigger, UrlExtractor, and FastText fallback are introduced through an additive component analysis, but the final model depends on their interaction. The addition of UrlExtractor seems especially useful as it is not affected by text length, mixed content or other content-based factors. The downside of the URL metadata method is that it relies on something that is not inherent to language. Thus, the UrlExtractor results require caution. URL metadata depends on whether the domain or path contains meaningful language hints, and these hints are not available or reliable for every document. And we have The Always English baseline to ensure that it actually shows intelligent design instead of targeting merely the largest class, as the English fallback can otherwise inflate the apparent strength of metadata-based prediction.

The runtime results narrow the practical claim. Selective routing does not automatically make the adapted systems faster than FastText, because runtime depends on how many documents trigger fallback. The full combined language-aware system gives the strongest adapted performance on OWI Frisian, while the split-trigger variant is cheaper but less accurate. The adaptation therefore offers a trade-off between recovery of misclassified rows and additional processing cost. The split-trigger variant appears to offer the best improvement without adding too much runtime. However, the actual trade-off cannot be decided outside of the full OWI indexing pipeline. As the datasets here either do not fully reflect the real setting or lack sample size.

Chapter 4

Related Work

Language identification is a common preprocessing step in multilingual text processing. In clean benchmark settings, especially with longer documents and high-resource languages, it is often treated as a largely solved task. This assumption becomes less reliable once the input changes. Short fragments, noisy extraction, low-resource languages, closely related language pairs, and web-crawled data all make the task less stable.

This chapter positions the thesis within that gap. The goal of this work is not to propose a new general-purpose language identification model. Instead, we examine whether the language identification method used in Resiliparse remains reliable under Open Web Index conditions, with Frisian as the main low-resource case. The related work therefore serves three purposes: it explains why profile-based language identification is still relevant in web pipelines, why low-resource and closely related languages remain difficult, and why web-domain evaluation is necessary alongside cleaner benchmark evaluation.

4.1 Language Identification Methods

Early and still influential language identification methods rely on character n-gram profiles. Cavnar and Trenkle introduced a profile-ranking method in which a document profile is compared against language profiles using an out-of-place ranking score [6]. This type of method remains relevant because it is simple, fast, and suitable for large-scale processing. These properties are important for web indexing, where language identification must be applied to large numbers of documents.

Survey work by Jauhiainen et al. shows that language identification should not be understood as one uniform task [9]. Performance depends on factors such as document length, domain, writing system, label granularity, and language relatedness. This is central to the present thesis. We do not only ask whether Resiliparse can identify languages in general, but whether its performance remains stable under the specific conditions of web-extracted OWI data.

FastText-style classifiers occupy a different position in the same design space. They use supervised learning with subword information and are widely used because they remain efficient while often outperforming older profile-based systems [11, 10]. In this thesis, Resiliparse and FastText are therefore not treated simply as interchangeable classifiers. Resiliparse represents the fast profile-based first pass used in the web pipeline, while FastText represents a stronger fallback option. The practical question is whether FastText can be used selectively, rather than replacing the fast first pass entirely.

4.2 Short Text, Web Noise, and Domain Mismatch

A recurring finding in language identification research is that performance depends on the amount and quality of available text. Baldwin and Lui show that language identification accuracy declines on short text, meaning that results obtained on longer documents do not necessarily transfer to short fragments [3]. Vatanen et al. similarly show that reduced input length leaves less linguistic evidence for reliable classification with n-gram models [22]. These findings are directly relevant to web-extracted data, where the remaining text after extraction may be short, fragmented, or partly unrelated to the main language of the page.

The web setting adds further problems. Extracted pages may contain boilerplate, navigation text, template fragments, advertisements, duplicated material, or mixed-language content. This means that language identification in a web-indexing pipeline differs from language identification on clean reference documents. Sch"afer's work on boilerplate detection is relevant here because the language identifier usually operates after extraction, and extraction quality determines how much useful linguistic signal remains [18].

Kreutzer et al. show that multilingual web-crawled datasets can contain severe language-label noise, low-quality text, and uneven quality across languages [13]. This supports one of the assumptions behind this thesis: language labels produced during crawling or indexing should not be treated as ground truth without validation. For that reason, the OWI slices used in this research are manually annotated before being used as evaluation data.

This literature motivates the benchmark-deployment gap examined in the thesis. Clean benchmark data remains useful as a reference point, but it does not fully represent the conditions encountered in a web index. We therefore compare Resiliparse across cleaner benchmark settings and noisier web-facing settings, rather than relying on a single evaluation environment.

4.3 Low-Resource and Closely Related Languages

Low-resource language identification introduces an additional difficulty. Many languages have limited clean training data, limited labelled web data, and weaker coverage in standard benchmarks. This can lead to weaker language profiles, sparse evaluation data, and confusion with better-resourced neighbouring languages.

However, the term *low-resource* is not governed by a single definition. Nigatu et al. argue that resource status is multidimensional: a language may have speakers or some forms of digital content while still lacking the labelled datasets, tools, institutional support, or task-specific resources needed for a particular area of language technology [15]. A language can therefore be relatively well resourced for one task and poorly resourced for another. This makes it necessary to state the measurement used in an empirical comparison instead of treating the category as self-evident.

For this thesis, Wikipedia article count is used as a proxy for the availability of written digital text. This is task-specific choice as Wikipedia-derived corpora and benchmarks are common in written language identification, including WiLI-2018, and because the evaluated profile-based and supervised detectors depend on the availability of written text.

Recent work has increasingly focused on these problems. GlotLID targets broad low-resource coverage and discusses challenges such as noisy data, high-resource leakage, and confusion between closely related languages [12]. AfroLID makes a similar argument for African languages, showing that reliable language identification is itself necessary for building useful language resources and that manually curated multi-domain data is important when existing tools are unreliable [1]. OpenLID and OpenLID-v3 also show that strong LID systems can still struggle with noise and closely related languages in web-derived settings [5, 8].

This body of work motivates the focus on Frisian. Frisian is not only lower-resource in the operational setting of this thesis, but also closely related to Dutch and other Germanic languages. Poor performance on Frisian can therefore have several causes: limited language resources, similarity to neighbouring high-resource languages, noisy extracted text, domain mismatch, or a combination of these factors.

For that reason, this thesis does not treat low-resource LID as a single accuracy problem. The error analysis examines whether misclassifications are connected to text length, web noise, linguistic similarity, and Resiliparse's internal ranking behaviour. This allows us to ask whether errors are random failures, or whether the correct language is often still near the top of the ranking but not separated strongly enough from related alternatives.

4.4 Benchmarks and Web-Domain Evaluation

Benchmark datasets are still useful because they provide controlled reference points. WiLI-2018 provides Wikipedia-derived labelled text across many languages [20, 21]. In this thesis, it is useful because it allows Resiliparse's behaviour on cleaner language data to be compared with its behaviour on noisier web-extracted data. However, because WiLI-2018 is derived from Wikipedia, it is closer to the kind of clean data often used for language resources than to the heterogeneous text encountered in a web index.

Recent work has therefore placed more emphasis on web-domain evaluation. CommonLID is especially relevant because it is a human-annotated benchmark covering 109 languages and is designed to evaluate language identification under noisier and more heterogeneous web conditions [19]. Its motivation aligns closely with this thesis: clean benchmarks may overestimate how reliable language identification systems are when they are applied to real web text.

The experimental design of this thesis follows this distinction. WiLI-2018 is used as a cleaner reference point, while CommonLID and the OWI slices are used to examine behaviour under more realistic web-facing conditions. This allows us to test whether Resiliparse's performance is stable across environments, or whether it drops once the input resembles the intended deployment setting.

4.5 Metadata and Lightweight Adaptation

Web pages contain information beyond their extracted body text. URLs, domains, paths, and country-code top-level domains can sometimes provide additional language evidence. Baykan et al. show that URLs can contain useful signals for web page language classification, especially when page content is unavailable or expensive to process [4]. This makes metadata relevant for web-specific LID.

However, URL evidence is not the same as text-based language identification. A URL may reflect domain ownership, country targeting, site structure, default language assumptions, or historical routing rather than the actual language of a page. For that reason, this thesis treats UrlExtractor as auxiliary evidence. It is evaluated both as a standalone metadata-based signal and as part of adapted systems that decide when Resiliparse should be trusted and when a fallback should be used.

This connects to the practical constraint of the thesis. The goal is not simply to maximise accuracy regardless of cost. In the OWI setting, language identification must remain fast enough for web-scale processing. Lightweight triggers are therefore important because they offer a way to improve reliability without applying a stronger detector to every document. The adapted systems

in this thesis are designed around this trade-off: keep Resiliparse when its output appears stable, and consult additional evidence when its ranking suggests ambiguity.

4.6 Position of This Thesis

The literature reviewed in this chapter establishes four points that frame the thesis. First, language identification is well studied, but its difficulty depends heavily on input length, domain, noise, language relatedness, and resource availability. Second, low-resource and closely related languages remain vulnerable, especially in noisy or web-derived settings. Third, clean benchmark performance does not necessarily reflect deployment performance in a web-indexing pipeline. Fourth, practical web-scale systems must balance accuracy against computational cost.

This thesis contributes to the space between these findings. Rather than proposing a new general-purpose LID model, it evaluates a concrete language identification component used in a web-oriented pipeline. It examines how Resiliparse behaves under OWI-style conditions, how its errors relate to web noise and ranking uncertainty, and whether selective fallback mechanisms can improve low-resource performance while preserving the motivation for a fast first-pass detector.

The related work therefore leads directly to the research design. Existing work explains why short, noisy, low-resource, and closely related language identification is difficult. This thesis examines how those difficulties appear in the specific context of the Open Web Index, and whether targeted adaptation can reduce them without abandoning web-scale efficiency.

Chapter 5

Conclusions

Through the evaluation of Resiliparse and the adapted fallback systems, we examined the reliability of Resiliparse's language identification under web-extracted conditions, which factors contribute to misclassification, and whether adapted versions can improve low-resource LID without abandoning the efficiency constraint of the web-indexing pipeline.

5.1 Summary of Findings

Our findings show that Resiliparse's weakness on low-resource languages is not absolute, but the result of a combination of factors. On clean or structured data, Resiliparse performs well on both high-resource and low-resource languages, showing that Frisian can be classified correctly. On web-extracted OWI data, however, performance drops sharply, with accuracy on the Frisian OWI slice reaching only 47%. This contrast supports the central claim of the thesis: the main challenge is not low-resource language identification in isolation, but low-resource language identification under noisy web-scale conditions.

The manually annotated OWI slices also revealed an important data-quality issue. The Frisian slice was not a clean collection of Frisian documents: only 42% of the annotated rows were Frisian, while English and Dutch each accounted for 20%. This contamination helps explain why the OWI setting is harsher than the benchmark setting. The detectors not only are required to identify Frisian, but to correctly classify a low-resource language within language-indexed web slice that already contains substantial related language contamination.

The error analysis identified that in cases of language misclassification, the correct classification was often present in the top 3 scores of the detector. In the Germanic Language Cluster, incorrect predictions often occurred when the correct language was nearby in the ranking but separated by only a small OOP score gap. Cases where the second-ranked language was correct had a median OOP gap of only 5, compared to 78.0 for correct rank-1 predictions. This suggests that Resiliparse frequently has the correct language within reach, but lacks a sufficiently sensitive mechanism for recognising ambiguous cases.

The adapted systems show that this weakness can be reduced through selective fallback. The language-aware RP+FastText fallback increased Frisian accuracy from 47% to 85%, while the combined RP+FastText+URL language-aware system reached 87%. In contrast, the default cutoff alone did not recover the same errors. This shows that the fallback classifier is currently not enough by itself: the trigger must also identify the rows where Resiliparse is likely to be wrong. We should continue to consider for the current implementation that the training data differs from the data within the web indexing pipeline.

Beyond the detector evaluation itself, this thesis contributes a manually annotated dataset of web-extracted content. Human-annotated web language identification datasets remain relatively uncommon compared to benchmark corpora derived from sources such as Wikipedia. The OWI slices developed for this research therefore provide a useful resource for evaluating LID under conditions that more closely resemble real deployment.

5.2 Practical Implications for the OWI

Our findings suggest that the OWI's current efficiency-oriented pipeline can disproportionately affect low-resource languages when the language signal is fragile. As the OWS initiative is striving for broad and unbiased access to information, reliable classification of minority-language content seems like an integral part of the mission.

The success of the language-aware fallback shows that meaningful accuracy improvements can be achieved without replacing Resiliparse entirely. This is important for the OWI because the system must remain computationally plausible at web scale. The practical implication is therefore not that statistical LID should be abandoned, but that its fallback behaviour should be made more sensitive to uncertainty, especially when closely related languages compete in the ranking. Another consideration still would be to work towards web-extracted training data, as CommonLID is an evaluation only dataset.

Our metadata results lead to a more cautious conclusion. UrlExtractor performs strongly on the target slice and can provide useful web-specific evidence, but it should not be treated as a general replacement for content-based LID. URL and domain cues are uneven: sometimes they reinforce the language signal, while in other cases they are missing, ambiguous, or default to English. Their most useful role is therefore as auxiliary evidence when the text-based prediction is fragile.

More broadly, our results show that low-resource language representation is partly shaped by infrastructure-level decisions. Thresholds, fallback rules, training sources, and metadata handling all influence whether minority-language documents remain visible in the correct part of the index. Improving LID in this setting is therefore not only a matter of replacing the underlying model, but also of adapting the pipeline to recognise when additional evidence is needed.

5.3 Reflections on Validity and Limitations

Several limitations should be considered when interpreting these findings. The manually annotated OWI slices contain 300 documents in total and therefore represent only a small sample of the wider Open Web Index. Certain domains, content types, or language communities may be overrepresented. The results should therefore be interpreted as evidence from a targeted evaluation slice rather than as a complete measurement of the OWI crawl.

The sampling method also limits what can be concluded. The Frisian and Dutch slices were sampled from language-indexed crawl data, meaning that the evaluation primarily captures false positives within those language slices. Pages written in Frisian but indexed under another language category do not appear in the Frisian slice and could not be measured directly. Finding these false negatives remains outside the scope of this thesis.

The annotation process introduces further uncertainty. The slices were annotated by two annotators, and agreement was high, reaching 96% for the Frisian slice and 98% for the random slice. However, disagreements often involved closely related languages, mixed-language pages, dialectal variation, or cases where the correct label was inherently ambiguous. Google Translate was used to assist with languages outside the annotators' expertise, which made broader

annotation possible but may not capture subtle linguistic distinctions.

The interpretation of error factors is also observational rather than causal. Short text length, extraction noise, linguistic similarity, and small OOP gaps are associated with reduced performance, but the experiments do not isolate each factor perfectly. The claim is therefore not that linguistic similarity alone causes the errors, but that it is an observed source of ambiguity in the evaluated setting.

Finally, Frisian should be understood as a case study for low-resource language identification, not as a complete proxy for all low-resource languages. Some findings, such as the gap between benchmark and deployment performance, are likely to generalise beyond Frisian. Other findings, especially the strong Dutch-Frisian confusion pattern, are tied to the linguistic and web-indexing context of this particular case.

5.4 Recommendations and Future Work

Several directions emerge from the findings of this thesis.

First, the OWI initiative should consider similarity-aware routing for fragile language clusters. The results show that the relationship between the top-ranked Resiliparse candidates can be used as an uncertainty signal, allowing stronger fallback evidence to be reserved for ambiguous cases.

Second, future work should expand the Research CLI and evaluation methodology to additional language clusters, such as Romance or Slavic languages. This would help determine whether the rank-order ambiguity observed in the Germanic Language Cluster is a broader phenomenon or specific to this language family.

Third, future research should evaluate larger and more diverse samples of web-extracted data. In particular, analysing pages that were indexed under incorrect language categories would provide greater insight into false negatives and help determine how representative the observed Frisian failure modes are across the wider crawl. Future research should also attempt to create a larger and more reliable annotated web-extracted training dataset.

Fourth, the OWI initiative should explore alternative training sources that include more web-extracted content. Better-matched training data may improve the statistical profiles used by fast first-pass identifiers and reduce the need for fallback in some cases.

Finally, future work could investigate more refined routing mechanisms that incorporate metadata, confidence calibration, or adaptive thresholds. The current language-aware fallback systems show that selective fallback can be effective, but there remains room to improve when and how stronger LID models are consulted.

5.5 Overall Conclusion

Resiliparse's low-resource weakness is conditional rather than absolute. Frisian can be identified accurately in clean benchmark settings, but performance deteriorates substantially when the same task is placed in the context of web-extracted OWI data. The central issue is therefore not simply whether low-resource languages can be identified, but whether they can be identified reliably under noisy, heterogeneous, and computationally constrained web-scale conditions.

The results show that much of this weakness is connected to uncertainty. Short text, extraction noise, mixed-language content, and linguistic similarity reduce the evidence available to the statistical classifier. In these cases, Resiliparse may still rank the correct language nearby, but fail to separate it strongly enough from competing alternatives. This makes low-resource LID not only a classification problem, but also an uncertainty-handling problem.

The language-aware fallback demonstrates that this weakness can be reduced through targeted adaptation. By identifying cases of low language-distance and selectively consulting stronger fallback evidence, the system recovers much of the lost performance while preserving the principle of a fast statistical first pass. This suggests that the most practical path forward is not necessarily to replace Resiliparse wholesale, but to make its fallback decisions more sensitive to linguistic ambiguity and web-specific uncertainty.

Ultimately, this thesis shows that language identification should not be treated as a solved preprocessing step in web-scale indexing. For low-resource languages, reliability depends on how the system handles noisy input, ambiguous rankings, and closely related neighbouring languages. By adopting similarity-aware and deployment-conscious LID strategies, the Open Web Index can move closer to its goal of providing an inclusive, transparent, and representative gateway to the world's digital information.

Bibliography

[1] Ife Adebara, AbdelRahim Elmadany, Muhammad Abdul-Mageed, and Alcides Alcoba Inciarte. AfroLID: A Neural Language Identification Tool for African Languages. *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 1958–1981, 2022.

[2] Giuseppe Attardi. WikiExtractor. `https://github.com/attardi/wikiextractor`, 2026. Accessed: 2026-06-06.

[3] Timothy Baldwin and Marco Lui. Language Identification: The Long and the Short of the Matter. In *Proceedings of Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 229–237, 2010.

[4] Eda Baykan, Monika Henzinger, and Ingmar Weber. A Comprehensive Study of Techniques for URL-Based Web Page Language Classification. *ACM Transactions on the Web*, 7(1), 2013.

[5] Laurie Burchell, Alexandra Birch, Nikolay Bogoychev, and Kenneth Heafield. An Open Dataset and Model for Language Identification. *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2023.

[6] William B. Cavnar and John M. Trenkle. N-Gram-Based Text Categorization. In *Proceedings of SDAIR-94, 3rd Annual Symposium on Document Analysis and Information Retrieval*, pages 161–175, 1994.

[7] ChatNoir. Resiliparse Documentation: Language Detection. `https://resiliparse.chatnoir.eu/en/stable/man/parse/lang.html`, 2026. Accessed: 2026-06-05. Source code: `https://github.com/chatnoir-eu/chatnoir-resiliparse`.

[8] Mariia Fedorova, Nikolay Arefyev, Maja Buljan, Jindřich Helcl, Stephan Oepen, Egil Rønningstad, and Yves Scherrer. OpenLID-v3: Improving the Precision of Closely Related Language Identification – An Experience Report. *arXiv preprint arXiv:2602.13139*, 2026.

[9] Tommi Jauhiainen, Krister Lindén, and Heidi Jauhiainen. Language Identification in Texts: A Survey. *Journal of Artificial Intelligence Research*, 65:675–782, 2019.

[10] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hervé Jégou, and Tomas Mikolov. FastText.zip: Compressing Text Classification Models. *arXiv preprint arXiv:1612.03651*, 2016.

[11] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of Tricks for Efficient Text Classification. *arXiv preprint arXiv:1607.01759*, 2016.

[12] Amir Hossein Kargaran, Ayyoob Imani, François Yvon, and Hinrich Schütze. GlotLID: Language Identification for Low-Resource Languages. *Findings of the Association for Computational Linguistics: EMNLP 2023*, 2024.

[13] Julia Kreutzer, Isaac Caswell, Lisa Wang, Ahsan Wahab, Daan van Esch, Nasanbayar Ulzii-Orshikh, Allahsera Tapo, Nishant Subramani, and Artem Sokolov. Quality at a Glance: An Audit of Web-Crawled Multilingual Datasets. *Transactions of the Association for Computational Linguistics*, 2022.

[14] Meta-Wiki contributors. List of Wikipedias, 2026. Accessed 13 June 2026. Archived snapshot: `https://web.archive.org/web/20260613123537/https://meta.wikimedia.org/wiki/List_of_Wikipedias`.

[15] Hellina Hailu Nigatu, Atnafu Lambebo Tonja, Benjamin Rosman, Thamar Solorio, and Monojit Choudhury. The Zeno's paradox of 'low-resource' languages. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17753–17774, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[16] OpenWebSearch.eu. Open Web Index. `https://openwebsearch.eu/open-webindex/`, 2026. Accessed: 2026-06-04. Archived at: `https://web.archive.org/web/20260604183011/https://openwebsearch.eu/open-webindex/`.

[17] OpenWebSearch.eu. Project - OpenWebSearch.eu. `https://openwebsearch.eu/the-project/`, 2026. Accessed: 2026-06-04. Archived at: `https://web.archive.org/web/20260604182935/https://openwebsearch.eu/the-project/`.

[18] Roland Schäfer. Accurate and Efficient General-Purpose Boilerplate Detection for Crawled Web Corpora. *Language Resources and Evaluation*, 51(3):873–889, 2017.

[19] Pedro Ortiz Suarez, Laurie Burchell, Catherine Arnett, Rafael Mosquera-Gómez, Sara Hincapie-Monsalve, Thom Vaughan, Damian Stewart, Malte Ostendorff, Idris Abdulmumin, Vukosi Marivate, Shamsuddeen Hassan Muhammad, Atnafu Lambebo Tonja, Hend Al-Khalifa, Nadia Ghezaiel Hammouda, Verrah Otiende, Tack Hwa Wong, Jakhongir Saydaliev, Melika Nobakhtian, Muhammad Ravi Shulthan Habibi, Chalamalasetti Kranti, Carol Muchemi, Khang Nguyen, Faisal Muhammad Adam, Luis Frentzen Salim, Reem Alqifari, Cynthia Amol, Joseph Marvin Imperial, Ilker Kesen, Ahmad Mustafid, Pavel Stepachev, Leshem Choshen, David Anugraha, Hamada Nayel, Seid Muhie Yimam, Vallerie Alexandra Putra, My Chiffon Nguyen, Azmine Toushik Wasi, Gouthami Vadithya, Rob van der Goot, Lanwenn ar C'horr, Karan Dua, Andrew Yates, Mithil Bangera, Yeshil Bangera, Hitesh Laxmichand Patel, Shu Okabe, Fenal Ashokbhai Ilasariya, Dmitry Gaynullin, Genta Indra Winata, Yiyuan Li, Juan Pablo Martínez, Amit Agarwal, Ikhlasul Akmal Hanif, Raia Abu Ahmad, Esther Adenuga, Filbert Aurelian Tjiaranata, Weerayut Buaphet, Michael Anugraha, Sowmya Vajjala, Benjamin Rice, Azril Hafizi Amirudin, Jesujoba O. Alabi, Srikant Panda, Yassine Toughrai, Bruhan Kyomuhendo, Daniel Ruffinelli, Akshata A, Manuel Goulão, Ej Zhou, Ingrid Gabriela Franco Ramirez, Cristina Aggazzotti, Konstantin Dobler, Jun Kevin, Quentin Pagès, Nicholas Andrews, Nuhu Ibrahim, Mattes Ruckdeschel, Amr Keleg, Mike Zhang, Casper Muziri, Saron Samuel, Sotaro Takeshita, Kun Kerdthaisong, Luca Foppiano, Rasul Dent, Tommaso Green, Ahmad Mustapha Wali, Kamohelo Makaaka, Vicky Feliren, Inshirah Idris, Hande Celikkanat, Abdulhamid Abubakar, Jean

Maillard, Benoît Sagot, Thibault Clérice, Kenton Murray, and Sarah Luger. CommonLID: Re-evaluating State-of-the-Art Language Identification Performance on Web Data. *arXiv preprint arXiv:2601.18026*, 2026.

[20] Martin Thoma. The WiLI Benchmark Dataset for Written Language Identification. *arXiv preprint arXiv:1801.07779*, 2018.

[21] Martin Thoma. WiLI-2018 - Wikipedia Language Identification Database, 2018.

[22] Tommi Vatanen, Jaakko J. Väyrynen, and Sami Virpioja. Language Identification of Short Text Segments with N-Gram Models. In *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC 2010)*, 2010.

[23] Jeffrey Scott Vitter. Random Sampling with a Reservoir. *ACM Transactions on Mathematical Software*, 11(1):37–57, 1985.

[24] Wikimedia Foundation. Wikimedia Downloads. `https://dumps.wikimedia.org/`, 2026. Accessed: 2026-06-06.

Appendix A

Technical Implementation

This appendix documents the implementation artifacts needed to reproduce the dataset preparation, detector extraction, analysis sweeps, and publication visualisations used in this thesis. The appendix is not intended to duplicate the full source tree. Instead, it records the command-line interface, the critical scientific logic, and the data schemas used by the reproducibility workflow.

A.1 Source Code Availability

The full source code for the research pipeline is available in the project repository:

`https://github.com/Smartie1242/rsp_eval`

For reproducibility, the repository contains a root `README.md` with the complete research workflow and a `code/README.md` file with the full command reference. These files document environment setup, OWI Slice annotation preparation, the manual Label Studio correction gate, GLC balancing, detector output generation, analysis commands, and publication visual generation.

A.2 CLI and Package Structure

The reusable implementation is packaged under `code/rsp/`. Commands are executed from the `code/` directory with the form `python -m rsp.cli.<command>`. The command groups are:

- **OWI preparation**: `prepare_datasets`, `compare_annotations`, and `owi_preprocessing` create Label Studio tasks, compare annotation exports, and enrich corrected OWI Slice files.
- **GLC balancing**: `balance_glc` creates the reproducible balanced Germanic Language Cluster dataset used for controlled experiments.
- **Extraction**: `resiliparse_outputs` runs the detector adapters and writes normalized `rp_outputs.csv` files under `code/extracted/`.
- **Analysis**: `score_gap`, `cutoff_sweep`, and `text_length_sweep` compute the score-gap, cutoff-routing, and text-length analysis artifacts.
- **Rendering**: `publication_visuals` renders publication-ready tables and figures.

- **Orchestration**: `pipeline` groups the above stages, but does not replace manual Label Studio annotation and correction.

The package modules separate reusable responsibilities across dataset loading, detector adapters, prediction output writing, routing and sweep computation, publication rendering, and Label Studio input preparation. The main module files are `datasets.py`, `detectors.py`, `outputs.py`, `routing.py`, `publication_visuals.py`, and `preannotation.py`.

A.3 Critical Implementation Snippets

The following snippets show the core logic used in the thesis. Full source files are available in the repository. The snippets are organized around the reproducibility pipeline: how detector outputs are captured with precise timing, how annotation inputs are validated, how adapted routing strategies are defined, and how final metrics are computed.

A.3.1 Runtime Measurement and Output Capture

Reproducibility requires exact and auditable measurement of both predictions and execution time. Listing A.1 shows how detectors are called and measured: each detector call is wrapped with `perf_counter()` to capture millisecond-precision timing. This timing method is essential for fair speed comparisons and validating speedup claims. The unified CSV schema preserves detector ranks, OOP scores, UrlExtractor evidence, and component runtimes in a single normalized table. This schema enables fair cross-model comparison and ensures that later analysis commands (routing, metrics, visualization) work identically regardless of which detector combination is being evaluated.

```
def run_model(text, model, n_results):
    """Run one detector and return `(results, runtime_seconds)`."""
    start = perf_counter()
    if model == "rp":
        results = detect_resiliparse(text, n_results)
    elif model == "ft":
        results = detect_fasttext(text, n_results)
    elif model == "url":
        results = detect_url(text)
    else:
        raise ValueError(f"Unknown model: {model}")
    end = perf_counter()
    return results, end - start
```

Listing A.1: Timed model execution: Each detector call is timed from start to end with `perf_counter()` to capture exact runtime in seconds.

The unified detector-output CSV header combines document labels, metadata, text length, and per-component runtimes with detector results from all adapters. The core columns are:

```
label, text_length, runtime_rp, runtime_ft, runtime_url, rank_1_lang_rp, rank_1_oop_score,
                rank_1_lang_ft, rank_1_probs, lang_url, url_score, ...
```

Rank columns repeat for each of the configured result counts (default: 5). This normalized format decouples detector pipelines from analysis commands.

A.3.2 Input Validation and Dataset Balancing

Reproducible evaluation requires validated annotation inputs. Listing A.2 shows how double annotations are aligned by stable document identifiers and compared to measure inter-annotator agreement. This comparison step is essential for documenting annotation quality and disagreement patterns in the thesis results. Listing A.3 demonstrates the deterministic balancing strategy used to create the Germanic Language Cluster: a controlled experimental input that is downsampled per language to enable fair comparison across resource levels. The manifest file (created by this function) records the exact seed, strategy, and per-language counts so that the balanced dataset can be reproduced precisely.

```
def align_annotations(a1, a2):
    """Align annotations by stable doc_id when possible, otherwise by row order."""
    if all(item.get("doc_id") for item in a1 + a2):
        by_doc_1 = {item["doc_id"]: item for item in a1}
        by_doc_2 = {item["doc_id"]: item for item in a2}
        if set(by_doc_1) != set(by_doc_2):
            raise ValueError("Files have different doc_id sets")
        return [(by_doc_1[doc_id], by_doc_2[doc_id]) for doc_id in sorted(by_doc_1)]

    if len(a1) != len(a2):
        raise ValueError("Files have different number of samples")
    return list(zip(a1, a2))


def compare_annotations(a1, a2) -> ComparisonSummary:
    """Compare two normalized annotation lists."""
    aligned = align_annotations(a1, a2)
    agree = 0
    disagreements = []

    for x, y in aligned:
        if x["label"] == y["label"]:
            agree += 1
        else:
            disagreements.append(
                {
```

Listing A.2: Double-annotation alignment and agreement measurement: Compares Label Studio exports by document identifiers (when available) or row order, recording per-item disagreements. Agreement rate documents annotation quality for OWI Slice evaluation inputs.

```
        split_file = language_dir / f"{split}.txt"
        if not split_file.exists():
            continue
        with open(split_file, "r", encoding="utf-8") as handle:
            language_lines[language_dir.name] = [line for line in handle if line.strip()]
    return language_lines


```

```python
def balance_split(input_dir: Path, output_dir: Path, split: str, seed: int):
    """Balance one split by downsampling each language to the smallest count."""
    language_lines = read_split_lines(input_dir, split)
    if not language_lines:
        return {}, {}

    input_counts = {language: len(lines) for language, lines in language_lines.items()}
    target_count = min(input_counts.values())
    output_counts = {}

    for language, lines in language_lines.items():
        rng = random.Random(f"{seed}:{split}:{language}")
        sampled = rng.sample(lines, target_count)
        language_dir = ensure_dir(output_dir / language)
        with open(language_dir / f"{split}.txt", "w", encoding="utf-8") as handle:
            handle.writelines(sampled)
        output_counts[language] = len(sampled)

    return input_counts, output_counts


def balance_glc(input_dir, output_dir, seed=42, splits=None):
    """Balance a GLC dataset and write a manifest."""
    input_dir = Path(input_dir)
    output_dir = ensure_dir(Path(output_dir))
    splits = splits or ["train", "val", "test"]

    if not input_dir.exists() or not input_dir.is_dir():
        raise FileNotFoundError(f"GLC input directory not found: {input_dir}")

    manifest = {
        "input_dir": str(input_dir),
        "output_dir": str(output_dir),
        "seed": seed,
        "strategy": "downsample_each_split_to_smallest_language_count",
        "created_at": datetime.now(timezone.utc).isoformat(),
        "splits": {},
    }

    for split in splits:
        input_counts, output_counts = balance_split(input_dir, output_dir, split, seed)
        if input_counts:
            manifest["splits"][split] = {
                "input_counts": input_counts,
                "output_counts": output_counts,
                "target_count": min(output_counts.values()),
            }
```

```
56
57      manifest_path = output_dir / "balance_manifest.json"
58      with open(manifest_path, "w", encoding="utf-8") as handle:
59          json.dump(manifest, handle, indent=2, ensure_ascii=False)
60
61      return manifest
```

Listing A.3: Deterministic dataset balancing: Downsamples each language to fixed counts by split while recording seed and strategy. The manifest enables exact reproduction of the balanced dataset used in controlled experiments.

```
1  BEGIN {
2      srand(seed)
3  }
4
5  {
6      if (NR <= k) {
7          reservoir[NR] = $0
8      } else {
9          j = int(rand() * NR) + 1
10         if (j <= k) {
11             reservoir[j] = $0
12         }
13     }
14 }
15
16 END {
17     for (i = 1; i <= k; i++) {
18         print reservoir[i]
19     }
```

Listing A.4: Reservoir sampling: Select fixed-size OWI samples without loading all candidates into memory. Deterministic seeding ensures reproducible samples across reruns, critical for dataset versioning.

```
1  <View>
2    <Text name="text" value="$text"/>
3    <Header value="Document info" />
4    <View style="box-shadow: 2px 2px 5px #999;
5                 padding: 20px; margin-top: 2em;
6                 border-radius: 5px;">
7      <Header value="Choose language"/>
8      <Choices name="annotated_language" toName="text" choice="single">
9
10       <!-- EXISTING VALUES (DO NOT CHANGE) -->
11       <Choice value="fy" alias="Frisian" />
12       <Choice value="nl" alias="Dutch" />
13       <Choice value="de" alias="German" />
14       <Choice value="en" alias="English" />
15
16       <!-- Remaining languages -->
```

Listing A.5: Label Studio annotation template: Single-label language choice interface with controlled vocabulary. Templates are version-controlled to ensure consistent annotation semantics across multiple annotation rounds and datasets.

A.3.3 Routing Strategies and Adapted Systems

The routing code defines the adapted systems evaluated in the results. Reproducibility requires that adaptation methods are repeatable and their logic is transparent. Listing A.6 defines the character-length bins used in text-length analysis, with the 1200-character boundary corresponding to the 200-word threshold discussed in the results. Listing A.7 implements the core routing logic: when the Resiliparse confidence (OOP score) falls below the cutoff or when rank-1 and rank-2 are linguistically distant, the system falls back to UrlExtractor (if available) or FastText. This routing is deterministic and data-dependent, enabling exact reproduction of composite predictions.

```
LENGTH_BINS = [0, 300, 600, 1200, 2400, 4800, 100000]
LENGTH_BIN_LABELS = [
    "0-300",
    "300-600",
    "600-1200",
    "1200-2400",
    "2400-4800",
    "4800+",
]


def valid_language_mask(series):
    """Return a boolean mask for language tags that can be compared
        safely."""
```

Listing A.6: Character-length bins for text-length analysis: Divide documents into six ranges with boundaries at 300, 600, 1200, 2400, and 4800 characters. The 1200-character boundary aligns with the 200-word threshold used in results interpretation.

```
    df,
    cutoff=HYBRID_CUTOFF,
    lang_thresh=LANG_AWARE_DISTANCE_THRESHOLD,
    score_col="rank_1_oop_score",
):
    """Route fragile Resiliparse cases to UrlExtractor when it supports
        rank 1/2, else FastText."""
    import pandas as pd

    df = normalize_prediction_columns(df)
    rank_2 = df["rank_2_lang_rp"].fillna("unknown").astype(str)
    scores = pd.to_numeric(df[score_col], errors="coerce").fillna(float("
        inf"))
    url_scores = pd.to_numeric(df.get("url_score", 0.0), errors="coerce")
        .fillna(0.0)

```

```
    fallback = (scores >= cutoff) | rank_distance_trigger_mask(df,
        lang_thresh=lang_thresh)
    valid_url = valid_language_mask(df["lang_url"])
    explicit_url = url_scores > URL_ENGLISH_FALLBACK_SCORE
    url_supports_rank_1 = valid_url & explicit_url & (df["lang_url"] ==
        df["rank_1_lang_rp"])
    url_supports_rank_2 = valid_url & (df["lang_url"] == rank_2)

    use_url = fallback & (url_supports_rank_1 | url_supports_rank_2)
    use_fasttext = fallback & ~use_url

    prediction = df["rank_1_lang_rp"].copy()
    prediction.loc[use_url] = df.loc[use_url, "lang_url"]
    prediction.loc[use_fasttext] = df.loc[use_fasttext, "rank_1_lang_ft"]

    return pd.DataFrame(
        {
            "prediction": prediction,
            "fallback": fallback,
            "use_url": use_url,
            "use_fasttext": use_fasttext,
        },
        index=df.index,
```

Listing A.7: Composite routing strategy: Combines Resiliparse (primary), UrlExtractor (fallback for high-confidence URL evidence), and FastText (fallback for low-confidence Resiliparse). Falling back either when OOP score exceeds cutoff or when rank-1/rank-2 are linguistically distant enables similarity-aware adaptation.

```
def derive_language_similarity_from_rp_outputs(paths: dict[str, Path]):
    import pandas as pd

    rows = []
    frames = {}
    log("similarity", f"{len(paths)} configured rp_outputs file(s)")
    for dataset, path in paths.items():
        if not path.exists():
            log("skip", f"{dataset}: missing similarity source {path}")
            continue

        df = pd.read_csv(path)
        try:
            require_columns(df, REQUIRED_LANGUAGE_SIMILARITY_COLUMNS,
                path)
        except ValueError as exc:
            if "rank_2_lang_rp" in str(exc):
                log("skip", f"{dataset}: language similarity requires
                    rank_2_lang_rp")
            else:
                log("skip", f"{dataset}: incompatible similarity schema
                    ({exc})")
            continue

```

```
        log("similarity", dataset)
        df = filter_evaluation_rows(df, dataset)
        frames[dataset] = df

    frames = prepare_wili_eval_overlap_frames(frames)
    for dataset, df in frames.items():
        log("similarity", dataset)
        for record in df.itertuples(index=False):
            true_lang = str(getattr(record, "label", "unknown") or "
                unknown")
            rank_1_lang = str(getattr(record, "rank_1_lang_rp", "unknown"
                ) or "unknown")
            rank_2_lang = str(getattr(record, "rank_2_lang_rp", "unknown"
                ) or "unknown")
            rank_3_lang = str(getattr(record, "rank_3_lang_rp", "unknown"
                ) or "unknown")
            oop_score = getattr(record, "rank_1_oop_score", math.nan)
            rank_2_oop_score = getattr(record, "rank_2_oop_score", math.
                nan)
            rank_3_oop_score = getattr(record, "rank_3_oop_score", math.
                nan)
            oop_score_numeric = pd.to_numeric(oop_score, errors="coerce")
            rank_2_oop_score_numeric = pd.to_numeric(rank_2_oop_score,
                errors="coerce")
            rank_3_oop_score_numeric = pd.to_numeric(rank_3_oop_score,
                errors="coerce")
            oop_gap = rank_2_oop_score_numeric - oop_score_numeric
            rank_1_valid = language_tag_is_valid(rank_1_lang)
            rank_2_valid = language_tag_is_valid(rank_2_lang)
            invalid = not (rank_1_valid and rank_2_valid)
            distance = safe_tag_distance(rank_1_lang, rank_2_lang)
            true_family = language_family(true_lang)
            predicted_family = language_family(rank_1_lang)
            rank_1_family = language_family(rank_1_lang)
            rank_2_family = language_family(rank_2_lang)
            rank_3_family = language_family(rank_3_lang)
            rank_1_correct = true_lang == rank_1_lang
            rank_2_correct = true_lang == rank_2_lang
            rank_3_correct = true_lang == rank_3_lang
            if rank_1_correct:
                rank_correctness = "Rank 1 correct"
            elif rank_2_correct:
                rank_correctness = "Rank 2 correct"
            elif rank_3_correct:
                rank_correctness = "Rank 3 correct"
            else:
                rank_correctness = "Neither rank correct"
```

Listing A.8: Language-similarity diagnostics: For each document, extract rank-1 and rank-2 predictions, compute their language distance, and compute the OOP score gap. Records whether rank-1, rank-2, or neither is correct. This normalized table enables analysis of prediction confidence relative to linguistic similarity.

A.3.4 Metrics Computation and Runtime Analysis

Publication tables are derived from normalized confusion counts (true/predicted pairs) and component timing data. This derivation is deterministic and reproducible: given the same confusion counts and timing data, the same tables are produced. Listing A.9 shows per-language metric calculation: for each language in each dataset, it computes precision, recall, and F1 from true positives, false positives, and false negatives. Listing A.10 shows how composite model runtimes are calculated by summing component runtimes only for components that are actually invoked by the routing strategy, enabling fair comparison of speed versus accuracy trade-offs.

```
            pred_mask = group["predicted_label"] == language
            tp = int(group.loc[true_mask & pred_mask, "count"].sum())
            fp = int(group.loc[~true_mask & pred_mask, "count"].sum())
            fn = int(group.loc[true_mask & ~pred_mask, "count"].sum())
            tn = total - tp - fp - fn
            support = tp + fn
            if support == 0:
                continue

            precision = tp / (tp + fp) if (tp + fp) else 0.0
            recall = tp / (tp + fn) if (tp + fn) else 0.0
            f1 = 2 * precision * recall / (precision + recall) if (
                precision + recall) else 0.0
            accuracy = (tp + tn) / total if total else 0.0
            rows.append(
                {
                    "dataset": dataset,
                    "model": model,
                    "language": language,
                    "language_family": language_family(language),
                    "resource_level": resource_level(language),
                    "accuracy": accuracy,
                    "precision_macro": precision,
                    "recall_macro": recall,
                    "f1_macro": f1,
                    "f1_weighted": f1,
                    "precision_weighted": precision,
                    "recall_weighted": recall,
                    "support": support,
                    "n_languages_dataset": len(labels),
                }
            )

    metrics_df = pd.DataFrame(rows)
    log("metrics", f"rows: {len(metrics_df):,}")
    return metrics_df


def model_metric_rows_from_confusion(confusion_df):
    """Compute one multiclass metric row per dataset/model from count
        data."""
    rows = []
    for (dataset, model), group in confusion_df.groupby(["dataset", "
        model"], sort=True):
```

Listing A.9: Per-language metric derivation: For each language, compute TP/FP/FN from confusion counts, then derive precision, recall, F1, and accuracy. Resource level is derived from article counts; language family from linguistic metadata. These metrics are the source for publication tables.

```
    import pandas as pd

    runtime_rp = pd.to_numeric(df.get("runtime_rp", 0), errors="coerce").
        fillna(0.0)
    runtime_ft = pd.to_numeric(df.get("runtime_ft", 0), errors="coerce").
        fillna(0.0)
    runtime_url = pd.to_numeric(df.get("runtime_url", 0), errors="coerce"
        ).fillna(0.0)

    if model == "Resiliparse":
        return runtime_rp
    if model == "FastText":
        return runtime_ft
    if model == "URL":
        return runtime_url

    scores = pd.to_numeric(df["rank_1_oop_score"], errors="coerce").
        fillna(math.inf)
    cutoff_fallback = scores >= cutoff
    if model == "RP+FastText":
        fallback = cutoff_fallback
        return runtime_rp + pd.Series(np.where(fallback, runtime_ft, 0.0)
            , index=df.index)
    if model == "RP+URL":
        fallback = cutoff_fallback
        return runtime_rp + pd.Series(np.where(fallback, runtime_url,
            0.0), index=df.index)
    if model == "RP+FastText lang-aware":
        fallback = cutoff_fallback | rank_distance_trigger_mask(df)
        return runtime_rp + pd.Series(np.where(fallback, runtime_ft, 0.0)
            , index=df.index)
    if model == "RP+URL lang-aware":
        fallback = cutoff_fallback | rank_distance_trigger_mask(df)
        return runtime_rp + pd.Series(np.where(fallback, runtime_url,
            0.0), index=df.index)
    if model == "RP+FastText+URL lang-aware":
        decision = combined_url_fasttext_decision(df, cutoff)
        return (
            runtime_rp
            + pd.Series(np.where(decision["fallback"], runtime_url, 0.0),
                index=df.index)
            + pd.Series(np.where(decision["use_fasttext"], runtime_ft,
                0.0), index=df.index)
        )
    if model == "RP+FastText+URL split-trigger":
        decision = split_trigger_url_fasttext_decision(df, cutoff)
        return (
```

```
            runtime_rp
            + pd.Series(np.where(decision["use_url"], runtime_url, 0.0),
                index=df.index)
            + pd.Series(np.where(decision["use_fasttext"], runtime_ft,
                0.0), index=df.index)
        )
    raise ValueError(f"Unknown runtime model: {model}")
```

Listing A.10: Composite model runtime calculation: Sum component runtimes (Resiliparse, FastText, UrlExtractor) only for components invoked by the routing decision. This enables fair runtime comparison: baseline models pay only their own cost, while composite models pay only fallback costs when fallback is triggered.

A.4 Data Schemas

The implementation uses filesystem-backed JSON, JSONL, plain-text, and CSV files.

A.4.1 OWI Slice Files

Each OWI Slice lives under `code/data/OWI_slice/<slice>/`, where `<slice>` is `frisian`, `dutch`, or `random`. The main files are:

File	Purpose
`raw.jsonl`	Raw sampled OWI documents before annotation preparation. Each row represents one candidate web document.
`cleaned.json`	Cleaned documents after preprocessing and filtering for Label Studio.
`labelstudio.json`	Label Studio import tasks with document text and metadata.
`annotations/*.json`	Label Studio exports from individual annotators. Current double annotations exist for Frisian and random slices; Dutch has only Marten annotations.
`diff.json`	Disagreement report for slices with two annotators.
`corrected.json`	Manually resolved annotations required before enrichment and OWI-based result generation.
`enriched.json`	Final enriched OWI Slice evaluation input, produced from corrected annotations.

Table A.1: OWI Slice file roles in the annotation and enrichment workflow.

A.4.2 OWI Slice Language Composition

The OWI slice language-composition table is included in Section 3.3.2, because it is part of the dataset description rather than only supplementary material. Table 3.1 records the filtered single-label composition used in detector evaluation.

A.4.3 GLC Layout

The Germanic Language Cluster uses the Resiliparse language dataset layout. Direct output is stored under `code/data/GLC/`; the deterministic balanced version is stored under `code/data/GLC_balanced/`. Both layouts use one directory per language and one text file per split:

```
data/GLC/<language>/train.txt
data/GLC/<language>/val.txt
data/GLC/<language>/test.txt

data/GLC_balanced/<language>/train.txt
data/GLC_balanced/<language>/val.txt
data/GLC_balanced/<language>/test.txt
data/GLC_balanced/balance_manifest.json
```

Listing A.11: GLC directory layout.

The balancing manifest records the seed, strategy, timestamp, input counts, and output counts so that the balanced experiment input can be reproduced.

A.4.4 Detector Output CSV

Detector extraction writes one `rp_outputs.csv` file per dataset under `code/extracted/`. This CSV is the shared input for the analysis and publication-visual commands. The CSV schema preserves detector ranks, OOP scores, runtimes, and evidence sources in a single normalized table. The core column groups are:

Column group	Description
Document fields	Dataset row identifiers, the gold `label`, source text, URL where available, and `text_length`.
Resiliparse ranks	Rank columns such as `rank_1_lang_rp`, `rank_2_lang_rp`, and associated OOP score columns such as `rank_1_oop_score`.
FastText prediction	FastText top prediction columns such as `rank_1_lang_ft` and confidence/score fields where available.
UrlExtractor prediction	UrlExtractor language prediction in `lang_url`, when URL evidence is available.
Runtime fields	Per-component timing fields such as `runtime_rp`, `runtime_ft`, and `runtime_url`. Composite runtime calculations assume routing itself is free and add only the component runtimes used.

Table A.2: Practical schema groups in `rp_outputs.csv`.

A.4.5 Publication Visual Inputs

Publication rendering normalizes detector outputs into standardized table shapes. The normalized metrics table captures per-language accuracy and F1 scores grouped by resource level and language family. The header row is:

```
dataset, model, language, language_family, resource_level, accuracy, precision_macro,
  recall_macro, f1_macro, f1_weighted, precision_weighted, recall_weighted, support
```

The normalized confusion table records aggregate true/predicted label pairs used for metrics computation:

```
dataset, model, true_label, predicted_label, count
```

Language-similarity visuals derive an additional normalized table with true language, rank-1 and rank-2 language predictions, language-family fields, rank correctness, rank-1/rank-2 distance, rank-1 and rank-2 OOP scores, and the rank-2-minus-rank-1 OOP score gap. The distance is computed between Resiliparse rank 1 and rank 2, while correctness records whether the true label appears at rank 1, rank 2, or neither rank.

Appendix B

Supplementary Results

The main results chapter includes the figures and tables needed for the central argument. This section records additional generated outputs that are useful for comparison, but that are not treated as primary evidence. The supplementary figures are included to make the analysis more transparent without expanding the main results section into a complete output catalogue.

B.1 Reference Tables

The main results chapter uses split tables close to the claims they support. Tables B.1, B.2, and B.3 preserve the fuller generated reference tables.

Table B.1: Supplementary resource-level baseline table.

Dataset	Model	Resource	Accuracy	Precision (macro)	Recall (macro)	F1 (macro)	F1 (weighted)	Languages
CommonLID	FastText	high	0.976	0.637	**0.969**	0.713	**0.819**	15
	FastText	low	0.962	0.330	0.291	0.309	0.030	3
	FastText	mid	0.956	**0.700**	0.882	**0.733**	0.767	7
	FastText	unknown	0.971	0.343	0.338	0.304	0.550	84
	RP+FastText	high	0.976	0.596	0.847	0.650	0.789	15
	RP+FastText	low	0.979	0.619	0.478	0.540	0.482	3
	RP+FastText	mid	0.937	0.628	0.809	0.657	0.696	7
	RP+FastText	unknown	0.967	0.296	0.285	0.243	0.450	84
	Resiliparse	high	0.974	0.566	0.757	0.600	0.747	15
	Resiliparse	low	**0.979**	0.611	0.474	0.533	0.490	3
	Resiliparse	mid	0.935	0.556	0.733	0.596	0.670	7
	Resiliparse	unknown	0.967	0.198	0.260	0.210	0.424	84
GLC	FastText	high	**0.997**	0.989	**0.999**	**0.994**	**0.994**	3
	FastText	low	0.991	**1.000**	0.962	0.981	0.981	1
	RP+FastText	high	0.992	0.994	0.973	0.983	0.983	3
	RP+FastText	low	0.989	0.997	0.959	0.977	0.977	1
	Resiliparse	high	0.992	0.994	0.973	0.983	0.983	3
	Resiliparse	low	0.989	0.997	0.959	0.977	0.977	1
OWI_slice_dutch	FastText	high	**0.976**	0.567	**0.607**	**0.586**	**0.961**	3
	RP+FastText	high	0.846	0.533	0.441	0.483	0.867	3
	Resiliparse	high	0.847	**0.583**	0.441	0.501	0.871	3
OWI_slice_frisian	FastText	high	0.984	0.905	**1.000**	0.947	0.959	4
	FastText	low	0.966	**1.000**	0.929	**0.963**	**0.963**	1
	FastText	unknown	**0.989**	0.000	0.000	0.000	0.000	1
	RP+FastText	high	0.815	0.688	0.158	0.253	0.266	4
	RP+FastText	low	0.494	0.486	0.810	0.607	0.607	1
	RP+FastText	unknown	**0.989**	0.000	0.000	0.000	0.000	1
	Resiliparse	high	0.815	0.688	0.158	0.253	0.266	4
	Resiliparse	low	0.494	0.486	0.810	0.607	0.607	1
	Resiliparse	unknown	**0.989**	0.000	0.000	0.000	0.000	1
OWI_slice_random	FastText	high	0.997	0.938	0.981	0.948	0.984	8
	FastText	mid	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1
	RP+FastText	high	0.978	0.931	0.870	0.867	0.931	8
	RP+FastText	mid	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1
	Resiliparse	high	0.978	0.931	0.870	0.867	0.931	8
	Resiliparse	mid	**1.000**	**1.000**	**1.000**	**1.000**	**1.000**	1
WiLI_Filtered	FastText	high	0.989	0.745	**0.984**	0.817	0.816	15
	FastText	low	0.993	0.499	0.489	0.494	0.494	2
	FastText	mid	0.996	0.854	0.965	0.887	0.887	7
	FastText	unknown	0.994	0.660	0.635	0.630	0.630	53
	RP+FastText	high	0.991	0.730	0.923	0.794	0.789	15
	RP+FastText	low	**0.998**	**0.888**	0.957	**0.921**	**0.921**	2
	RP+FastText	mid	0.993	0.771	0.931	0.811	0.811	7
	RP+FastText	unknown	0.992	0.504	0.551	0.519	0.519	53
	Resiliparse	high	0.991	0.730	0.923	0.794	0.789	15
	Resiliparse	low	**0.998**	**0.888**	0.957	**0.921**	**0.921**	2
	Resiliparse	mid	0.993	0.771	0.931	0.811	0.811	7
	Resiliparse	unknown	0.992	0.504	0.551	0.519	0.519	53

Table B.2: Supplementary full adapted-system summary across datasets.

Dataset	Model	Accuracy	Precision (macro)	Recall (macro)	F1 (macro)	F1 (weighted)	Support	Languages
CommonLID	Resiliparse	0.620	0.283	0.364	0.297	0.571	373230	109
	FastText	**0.681**	**0.406**	**0.459**	**0.388**	**0.629**	373230	109
	RP+FastText	0.652	0.368	0.401	0.334	0.600	373230	109
	RP+FastText lang-aware	0.677	0.398	0.456	0.382	0.626	373230	109
GLC	Resiliparse	0.970	**0.995**	0.970	0.982	0.982	68840	4
	FastText	**0.990**	0.992	**0.990**	**0.991**	**0.991**	68840	4
	RP+FastText	0.970	**0.995**	0.970	0.982	0.982	68840	4
	RP+FastText lang-aware	0.983	0.993	0.983	0.988	0.988	68840	4
OWI_slice_dutch	Resiliparse	0.791	**0.583**	0.441	0.501	0.871	86	3
	FastText	**0.965**	0.567	**0.607**	**0.586**	**0.961**	86	3
	RP+FastText	0.791	0.533	0.441	0.483	0.867	86	3
	RP+URL	0.791	0.533	0.441	0.483	0.867	86	3
	RP+FastText lang-aware	**0.965**	0.567	**0.607**	**0.586**	**0.961**	86	3
	RP+URL lang-aware	0.779	0.390	0.488	0.394	0.825	86	3
	RP+FastText+URL lang-aware	**0.965**	0.567	**0.607**	**0.586**	**0.961**	86	3
	RP+FastText+URL split-trigger	0.930	0.575	0.492	0.525	0.937	86	3
OWI_slice_frisian	Resiliparse	0.471	0.539	0.240	0.270	0.428	87	6
	FastText	**0.954**	0.770	**0.821**	**0.792**	**0.950**	87	6
	RP+FastText	0.471	0.539	0.240	0.270	0.428	87	6
	RP+URL	0.471	0.539	0.240	0.270	0.428	87	6
	RP+FastText lang-aware	0.851	0.785	0.755	0.763	0.856	87	6
	RP+URL lang-aware	0.862	0.797	0.750	0.765	0.866	87	6
	RP+FastText+URL lang-aware	0.874	**0.800**	0.763	0.775	0.879	87	6
	RP+FastText+URL split-trigger	0.782	0.786	0.692	0.719	0.784	87	6
OWI_slice_random	Resiliparse	0.900	0.939	0.884	0.882	0.932	90	9
	FastText	**0.978**	**0.944**	**0.983**	**0.954**	**0.984**	90	9
	RP+FastText	0.900	0.939	0.884	0.882	0.932	90	9
	RP+URL	0.900	0.939	0.884	0.882	0.932	90	9
	RP+FastText lang-aware	0.944	**0.944**	0.903	0.895	0.962	90	9
	RP+URL lang-aware	0.678	0.832	0.721	0.728	0.689	90	9
	RP+FastText+URL lang-aware	0.944	**0.944**	0.903	0.895	0.962	90	9
	RP+FastText+URL split-trigger	0.822	0.872	0.845	0.844	0.879	90	9
WiLI_Filtered	Resiliparse	0.672	0.582	0.668	0.610	0.611	39000	77
	FastText	**0.732**	**0.690**	**0.729**	**0.686**	**0.687**	39000	77
	RP+FastText	0.672	0.582	0.669	0.610	0.611	39000	77
	RP+FastText lang-aware	0.726	0.686	0.723	0.680	0.681	39000	77

Table B.3: Supplementary UrlExtractor and detector comparison for OWI Dutch and OWI random controls.

Dataset	Model	Accuracy	F1 (macro)	F1 (weighted)	Support	Mean runtime (ms)	Docs/s
OWI_slice_dutch	Always English	0.070	0.043	0.009	86	0.000	n/a
	URL	0.779	0.394	0.825	86	0.054	18682.2
	Resiliparse	0.791	0.501	0.871	86	0.243	4118.4
	FastText	0.965	0.586	0.961	86	3.686	271.3
	RP+URL	0.791	0.483	0.867	86	0.243	4110.9
	RP+URL lang-aware	0.779	0.394	0.825	86	0.296	3374.5
	RP+FastText+URL lang-aware	0.965	0.586	0.961	86	0.464	2156.4
	RP+FastText+URL split-trigger	0.930	0.525	0.937	86	0.288	3474.7
OWI_slice_random	Always English	0.356	0.058	0.187	90	0.000	n/a
	URL	0.678	0.679	0.678	90	0.036	27976.3
	Resiliparse	0.900	0.882	0.932	90	0.245	4082.6
	FastText	0.978	0.954	0.984	90	0.827	1209.2
	RP+URL	0.900	0.882	0.932	90	0.245	4082.6
	RP+URL lang-aware	0.678	0.728	0.689	90	0.279	3586.0
	RP+FastText+URL lang-aware	0.944	0.895	0.962	90	0.706	1417.2
	RP+FastText+URL split-trigger	0.822	0.844	0.879	90	0.262	3821.0

B.2 Resource-Level Supplement

Figure B.1 complements the macro-F1 resource plot in the main text with overall accuracy. This is useful for comparison because accuracy is easier to read operationally, while macro-F1 remains more central to the low-resource argument.

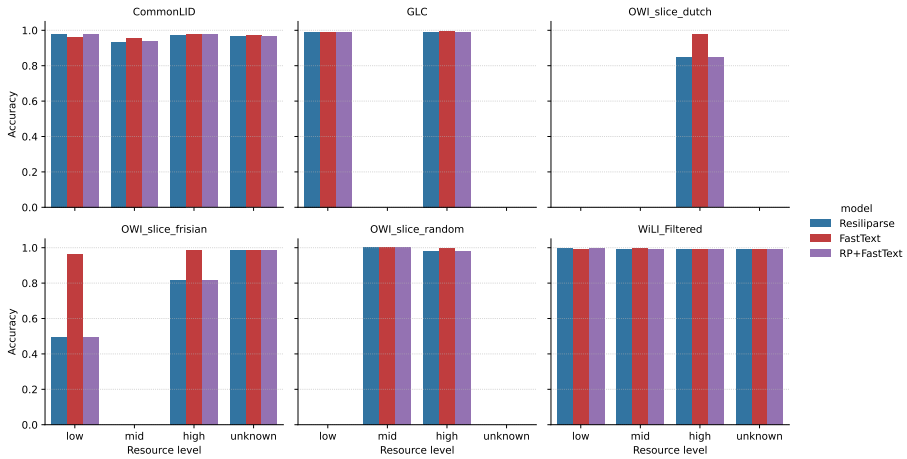

Figure B.1: Supplementary accuracy grouped by operational language resource level.

B.2.1 Text-Length Controls

The main text keeps the GLC and OWI Frisian text-length plots. Figures B.2 and B.3 provide the corresponding control plots for the other OWI slices and benchmark-style datasets.

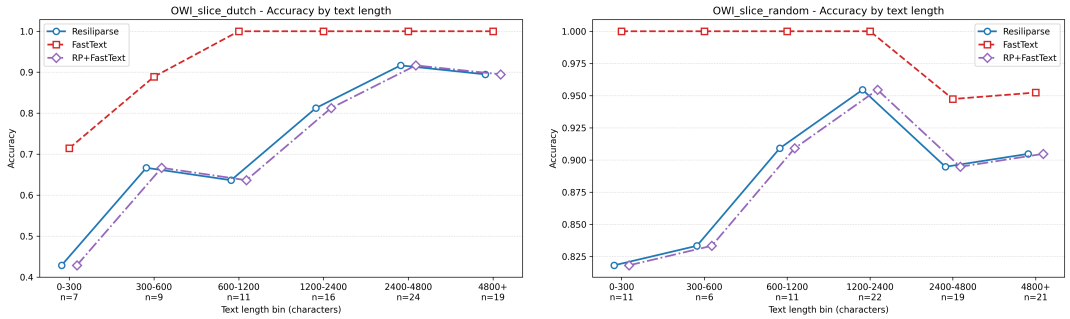

Figure B.2: Supplementary text-length accuracy controls for the OWI Dutch and random slices.

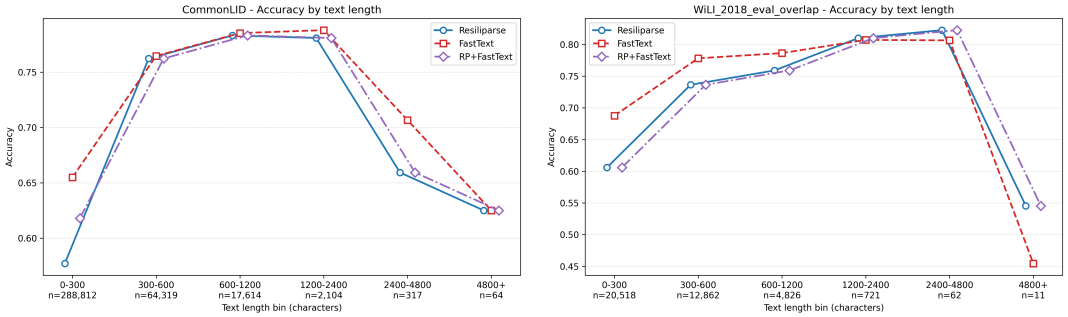

Figure B.3: Supplementary text-length accuracy controls for CommonLID and WiLI-2018.

B.3 Cutoff Sweep Controls

The main text presents cutoff-sweep results for GLC and the three OWI slices. CommonLID provides an additional comparison on a large multilingual benchmark and helps assess whether the observed routing behaviour generalises beyond the web-focused datasets.

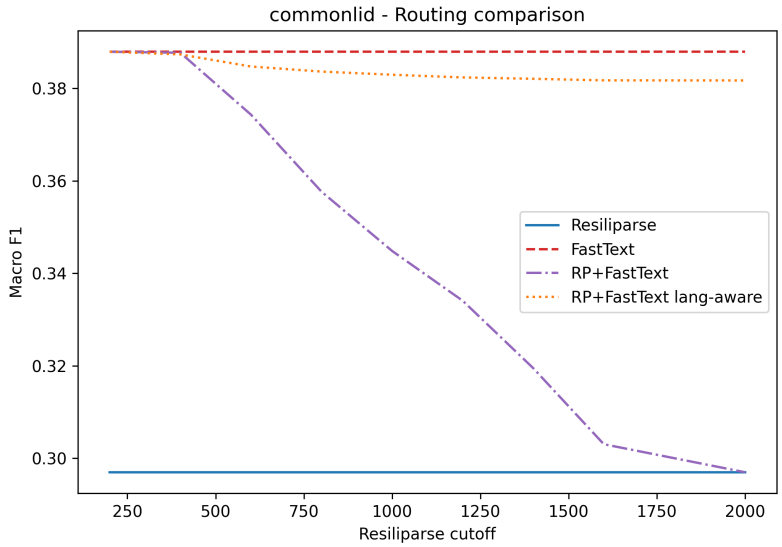

Figure B.4: Supplementary cutoff-sweep comparison for CommonLID.

B.4 Language-Similarity Diagnostics

The main text uses the OOP-gap correctness table as the clearest similarity evidence. The following diagnostics show alternative views of the same rank-distance and correctness signals.

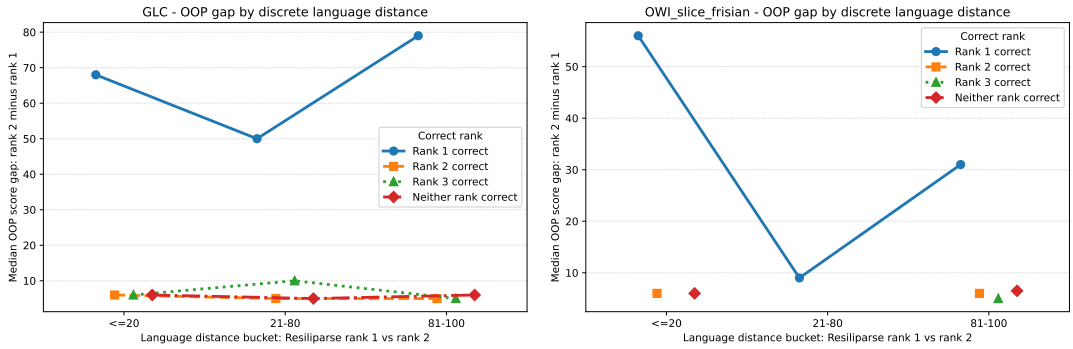

Figure B.5: Supplementary OOP-gap diagnostics by language-distance bin for GLC and OWI Frisian.

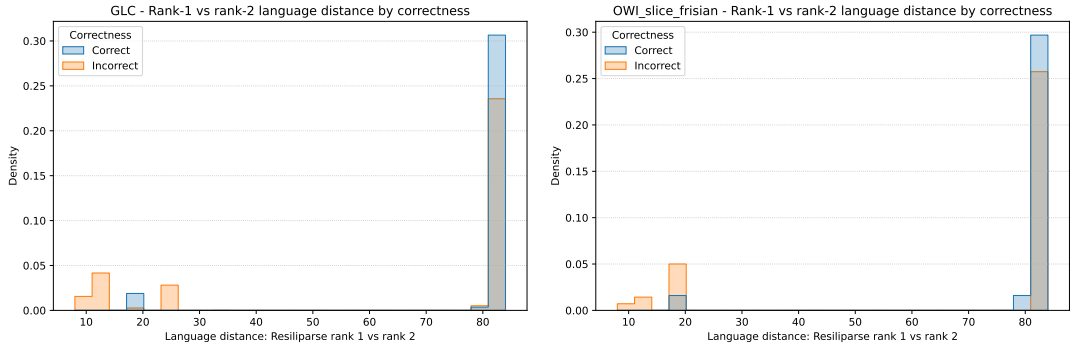

Figure B.6: Supplementary rank-1/rank-2 language-distance distributions by correctness for GLC and OWI Frisian.

Figure B.7: Supplementary family-level error flow for incorrect Resiliparse predictions on GLC.

B.5 Runtime Details

The results chapter reports speedups relative to Resiliparse and FastText. Table B.4 and Figure B.8 provide the absolute runtime summaries behind that comparison.

Table B.4: Supplementary absolute runtime summary for evaluated detector systems.

Dataset	Model	Mean (s)	Median (s)	P95 (s)	Total (s)	Docs/s	Documents
CommonLID	FastText	**0.0001**	0.0001	0.0002	35.87	10404.7	373230
	RP+FastText	0.0001	0.0001	0.0002	45.25	8248.5	373230
	RP+FastText lang-aware	0.0002	0.0002	0.0004	78.70	4742.2	373230
	Resiliparse	0.0001	0.0001	0.0002	43.37	8605.1	373230
GLC	FastText	0.0017	0.0007	0.0067	116.70	589.9	68840
	RP+FastText	0.0004	0.0002	0.0011	25.26	2725.2	68840
	RP+FastText lang-aware	0.0020	0.0009	0.0078	139.53	493.4	68840
	Resiliparse	**0.0004**	0.0002	0.0011	25.26	2725.3	68840
OWI_slice_dutch	FastText	0.0037	0.0005	0.0029	0.32	271.3	86
	RP+FastText	0.0002	0.0002	0.0005	0.02	4108.1	86
	RP+FastText lang-aware	0.0039	0.0007	0.0034	0.34	254.5	86
	RP+FastText+URL lang-aware	0.0005	0.0003	0.0010	0.04	2156.4	86
	RP+FastText+URL split-trigger	0.0003	0.0002	0.0006	0.02	3474.7	86
	RP+URL	0.0002	0.0002	0.0005	0.02	4110.9	86
	RP+URL lang-aware	0.0003	0.0002	0.0006	0.03	3374.5	86
	Resiliparse	0.0002	0.0002	0.0005	0.02	4118.4	86
	URL	**0.0001**	0.0000	0.0001	0.00	18682.2	86
OWI_slice_frisian	FastText	0.0005	0.0004	0.0012	0.04	2012.2	87
	RP+FastText	0.0002	0.0002	0.0003	0.02	5052.6	87
	RP+FastText lang-aware	0.0006	0.0005	0.0015	0.06	1558.0	87
	RP+FastText+URL lang-aware	0.0004	0.0003	0.0009	0.03	2826.2	87
	RP+FastText+URL split-trigger	0.0002	0.0002	0.0003	0.02	4486.9	87
	RP+URL	0.0002	0.0002	0.0003	0.02	5052.6	87
	RP+URL lang-aware	0.0002	0.0002	0.0003	0.02	4412.6	87
	Resiliparse	0.0002	0.0002	0.0003	0.02	5052.6	87
	URL	**0.0000**	0.0000	0.0000	0.00	30895.2	87
OWI_slice_random	FastText	0.0008	0.0005	0.0028	0.07	1209.2	90
	RP+FastText	0.0002	0.0002	0.0005	0.02	4082.6	90
	RP+FastText lang-aware	0.0010	0.0006	0.0032	0.09	971.6	90
	RP+FastText+URL lang-aware	0.0007	0.0003	0.0023	0.06	1417.2	90
	RP+FastText+URL split-trigger	0.0003	0.0002	0.0005	0.02	3821.0	90
	RP+URL	0.0002	0.0002	0.0005	0.02	4082.6	90
	RP+URL lang-aware	0.0003	0.0002	0.0005	0.03	3586.0	90
	Resiliparse	0.0002	0.0002	0.0005	0.02	4082.6	90
	URL	**0.0000**	0.0000	0.0001	0.00	27976.3	90
WiLI_Filtered	FastText	**0.0001**	0.0001	0.0003	5.25	7435.5	39000
	RP+FastText	0.0001	0.0001	0.0002	5.46	7147.7	39000
	RP+FastText lang-aware	0.0003	0.0002	0.0005	10.32	3780.4	39000
	Resiliparse	0.0001	0.0001	0.0002	5.45	7153.1	39000

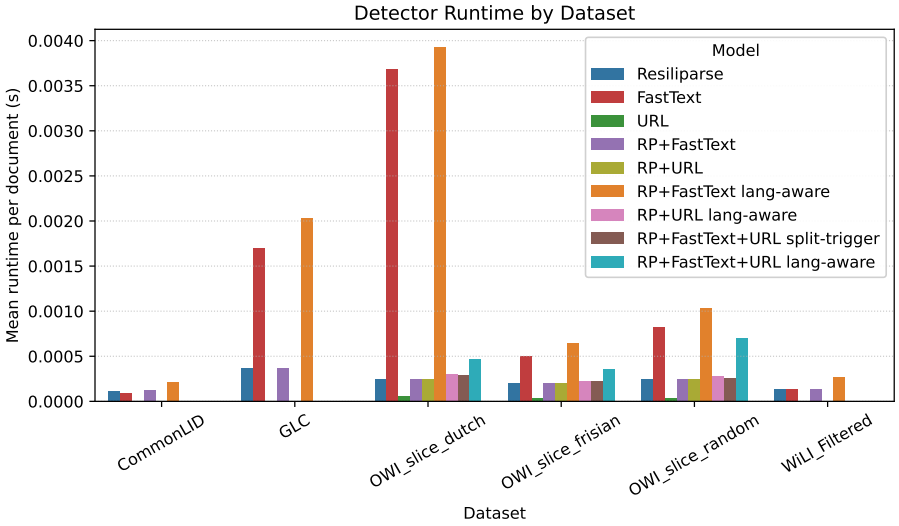

Figure B.8: Supplementary mean runtime by dataset and detector system.

B.6 Score-Gap Diagnostics

The score-gap command produces exploratory diagnostics for Resiliparse and FastText. Figure B.9 shows the controlled GLC summaries, while Figure B.10 shows the target OWI Frisian summaries.

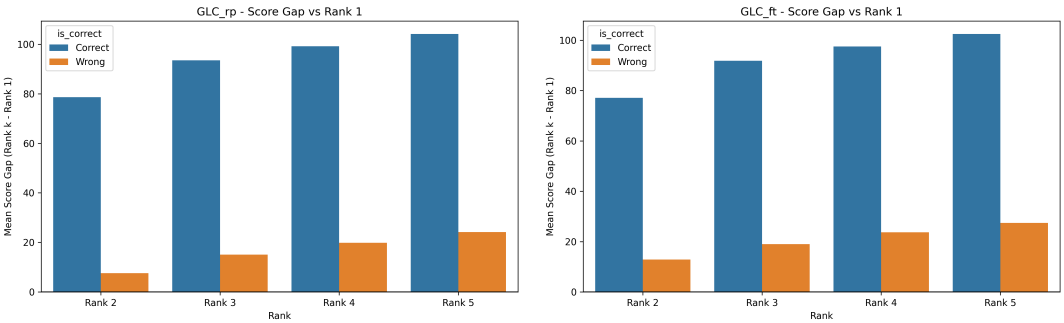

Figure B.9: Supplementary GLC score-gap summaries for Resiliparse and FastText.

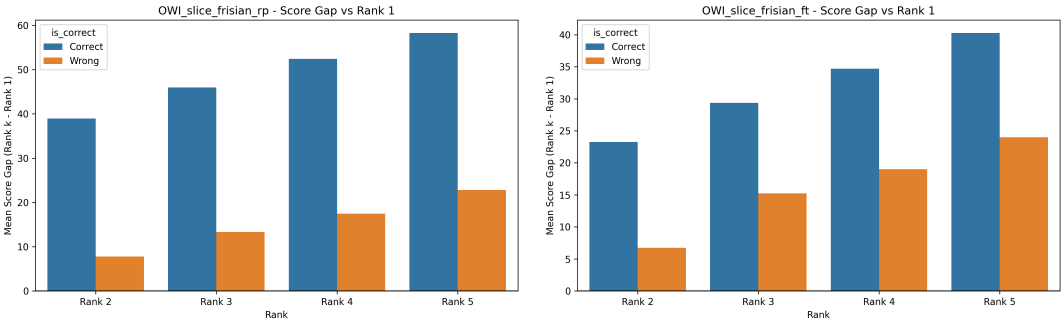

Figure B.10: Supplementary OWI Frisian score-gap summaries for Resiliparse and FastText.