University of Otago

School of Computing

COSC385 Project Report

---

# The Art of Reversed French

## Large Language Models Powered Verlan Identification

---

*Author:*
Yitian Li (4556502)

*Supervisor(s):*
Dr Lech Szymanski
Dr Veronica Liesaputra

October 23, 2025

**Abstract**

I need to rewrite this fucking thing

# 1 Introduction

## 1.1 Context and Motivation

Ever since the early nineteenth century, French-speaking people have employed verlan.[1] Like Pig Latin, verlan is an example of an *argot* created by reversing the syllables of a word [1, 2]. Today, verlan is commonly employed among adolescents and youth in francophone cultures[2] [3]. Some examples are:

- bonjour = bon + jour → jour + bon → jourbon (greetings)
- bite = bi + te → te + bi → tebie (penis)

In real-life conversations, such forms appear in sentences like the following:

- *Un p'tit*[3] *jourbon et tout le monde sourit.*
  (A quick hello and everyone smiles.)
- *Le graff géant représente une tebie pixel art.*
  (The giant graffiti depicts a pixel art penis.)

Verlan can also originate from other languages, such as English:

- shit = shi + t → t + shi → teuchi [3]
- *Il a du bon teuchi du bled.*
  (He's got some good shit from the countryside.)

In general, verlan obeys this syllable-flipping rule with only minimal accommodations for pronunciation (such as elision or addition of accents) [1]. Although slang has traditionally been performed more orally than in writing, internet use — especially texting and social media — now produces much

[1] In fact, the term *verlan* is itself a verlan conversion of *l'envers* ("the reverse").
[2] For example, France, Belgium, Switzerland, Luxembourg, and Canada.
[3] Standard spelling: petit.

written form; understanding how NLP systems handle them becomes increasingly important [4].

When speaking in more than one language, original slang terms can become an easy addition to text, making it difficult for machine translation algorithms to correctly understand them [5]. Translating English sentences into French using verlan, for instance, is a challenge that both Google Translate[4] and DeepL[5] face, despite both employing machine learning algorithms [6, 7]. For example, in the sentence *Le graff géant représente une tebie pixel art.*, Google Translate and (Figure 1) DeepL (Figure 2) also do not translate the word *tebie* well. More precisely, in DeepL, there is no desired translation, such as *penis* in its wordlist alternative (Figure 3).

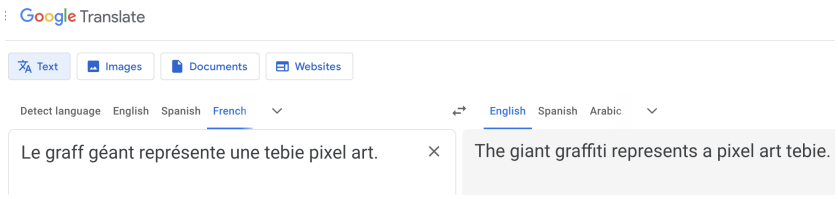



Figure 1: Google Translate cannot translate the verlan *tebie* correctly.

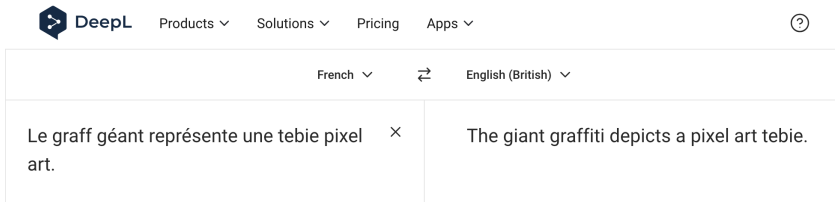



Figure 2: DeepL cannot translate the verlan *tebie* correctly.

[4] https://translate.google.com
[5] https://www.deepl.com

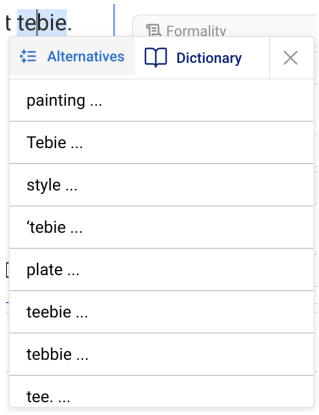



Figure 3: No desired translation for verlan *tebie* in DeepL's alternative word list.

These failures prompt an early attempt: automatic verlan detection in text. In reaction, this report examines how current machine learning models perform on the detection task. Previous works have tackled detecting slang with ML and translating noisy text more broadly [13, 8].

For verlan specifically, we are unaware of any publicly available, published computational research on its detection or translation (September 2025), nor is there a publicly available, curated dataset. The closest similar effort is a University of Toronto exercise whereby students are asked to train an NMT model to generate Pig Latin from English[6]; this educational exercise is in the opposite direction and does not attempt detection or normalisation.

Because translators these days never seem to do verlan quite correctly, clearly there is a need for targeted research into how machine learning models can detect verlan (with normalisation reserved for future work).

This document plugs that gap by developing datasets and models specifically aimed at verlan identification.

## 1.2 Objective

The primary goal of this project was originally to both *identify* and *normalise* verlan — that is, to detect verlan words in context and to convert them back to their standard French equivalents. To support these aims, we created a dictionary of verlan–standard French pairs for use in both identification

[6] https://uoft-csc413.github.io/2022/assets/assignments/PA03.pdf

and normalisation, as well as a sentence corpus illustrating these words in context. However, due to time constraints, this report focuses solely on the identification aspect; the normalisation task and its associated experiments are left for future work.

To enable this evaluation, we created targeted verlan resources — a lexicon of verlan terms and their standard French translations, and a sentence corpus demonstrating the position of these words in realistic contexts. The corpus accompanies each sentence with verlan matched against its standard French counterpart and also has tags to indicate whether a sentence is verlan. Together, these resources form an integrated dataset apt for rule-based matching and LLM-based training and evaluation. After that, the project embeds and classifies verlan using Large Language Models (LLMs) and analyses the results.

1. **Raw data collection** Gathering verlan terms and their standard French counterparts from online sources and existing dictionaries. This step focuses on scraping and compiling the raw linguistic material in advance of any processing.

2. **Dictionary creation** Cleaning and standardising the collected data into a structured lookup table (GazetteerEntries) that maps verlan words to their standard French equivalents.

3. **Sentence dataset creation** Building a corpus of example sentences illustrating verlan usage in context.

4. **LLM embedding** Encoding sentences using large language models to obtain numerical representations.

5. **Verlan classification** Training and evaluating classifiers on the embeddings to identify whether a sentence contains verlan.

6. **Results analysis** Interpreting model performance and identifying patterns or errors.

These steps collectively constitute the methodology centre of this study, providing a systematic underpinning to the following experiments.

The central aim of this report is to determine the degree to which machine learning models can pick up on verlan; the datasets were designed in order to be amenable to such assessment and provide a systematic test basis.

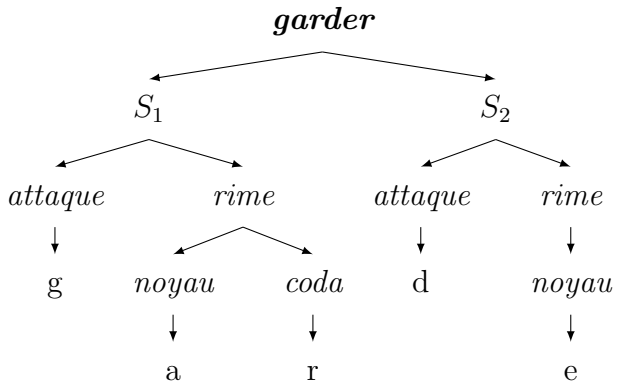
All the code and unannotated dataset for this project are available on GitHub[7]. The annotated, peer-reviewed dataset will be published shortly after this report, by the end of 2025.

# 2 Background

## 2.1 A Living Verlan

Méla [20] characterises verlan as a register inclined to seek opacity — used in contexts where speakers want to hide meaning. In order to maintain this opacity, speakers may use tactics such as *reverlanisation* (reversing a form once it has become familiar) and truncation. Verlan is not, however, random: it remains rule-governed, the key process being syllabic reversal (see Introduction).

Méla drew on the analytic model proposed by Kaye and Lowenstamm, adopting their syllable decomposition [21]. The syllable can be disassembled into *attaque* (onset), *rime* (rhyme), *noyau* (nucleus), and *coda*. For example, here is a representation of the word *garder*, IPA[8] [garde].



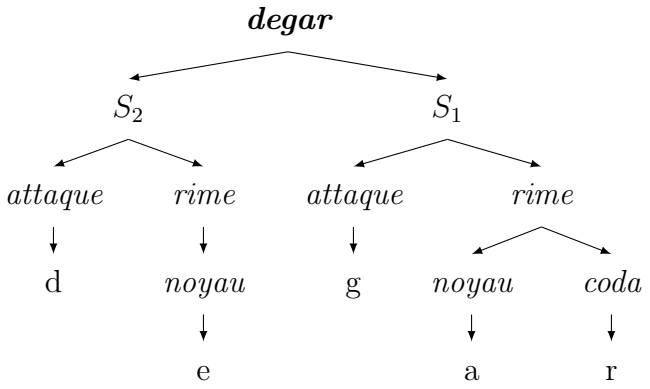

It has two syllables, $S_1$ and $S_2$. To create the verlan form, we follow the permutation equation below:

$$(S_1 S_2) \rightarrow (S_2 S_1) \tag{1}$$

[7] https://github.com/greateden/Verlan-Identification-Normalisation

[8] International Phonetic Alphabet, https://en.wikipedia.org/wiki/International_Phonetic_Alphabet

After the permutation, we obtain the verlan form of *garder* as *degar*, represented below.




Notably, this stylised example swaps only the syllable labels (i.e., between $S_1$ and $S_2$). In attested verlan, the swap is frequently accompanied by resyllabification, vowel adjustments, or segment loss (such as dropping a final *e*), so the internal structure of the syllables can change as well [20]. The example above should therefore be seen as an illustration rather than an exhaustive explanation of forming a verlan. For further details, readers are advised to consult Méla's paper[20].

Because verlan combines syllable-level permutation with these structural adjustments, it is difficult even for human readers [20]; therefore, it is sure to present challenges to current machine learning models. In this report, we use verlan as a case study for how non-standard subword permutations challenge both linguistic intuition and computational understanding. Like a mirror folded within itself, verlan reflects the creative tension between order and chaos — an echo of how language reinvents its own boundaries.

## 2.2 Detecting Slang

There is published work on verlan, but not much is specifically focused on detection. As of September 2025, we are not aware of published computational research specifically devoted to verlan detection. Available papers discuss verlan as part of larger slang datasets or corpora and do not focus on the detection task [9, 10, 11, 12].

Fortunately, there are several papers related to computational slang detection, and their approaches could contribute to verlan detection to a large extent[13, 14, 15, 17]. These studies address slang detection in general (not verlan specifically) and span multiple Indo-European languages (e.g., English, German, Russian).

Therefore, regarding the history of verlan detection, this report first generalises the task as slang detection, and then discusses possible methods that could be implemented for verlan identification, in order to provide readers with a general and useful background.

### 2.2.1 1910s-2010s: A Super-Condensed History of Slang Detection

The background of traditional slang detection historically used approximate (fuzzy) string matching. Two most commonly referenced techniques are: Soundex — a phonetic string-based English indexing method, developed by Odell and Russell (1918) — and Levenshtein's (1966) edit distance. The $\mathcal{O}(|s| \cdot |t|)$ time traditional dynamic-programming solution is described in Wagner and Fischer (1974). [22, 23, 48]

**How these methods work (high level).** *Soundex* translates an input token into an approximate phonetic key (first character + consonant class digits; consecutive duplicates folded; padded/truncated to fixed size). Two strings would match well if the keys match. It is biased toward matches with similar sounds even if there are minor differences in spelling, but it is extremely English-biased and would potentially have a high collision rate. *Levenshtein edit distance* is the minimum number of single-character insertions, deletions, or substitutions to change one string into another; the naive algorithm employs dynamic programming in $\mathcal{O}(|s| \cdot |t|)$ time. The distance is computed empirically by the Wagner-Fischer dynamic-programming algorithm in $\mathcal{O}(|s| \cdot |t|)$ time. Small distance (less than some cutoff) indicates similarity; the cutoffs are either absolute or length-normalised. [48]

**Performance and weaknesses for verlan/slang.** Phonetic keys typically yield high recall on name-like tokens but lower precision due to key collisions, and language-dependent rules do poorly on French slang. Edit distance handles typo-like noise and OCR well, but is not pronunciation sensitive and considers transpositions/syllable flips as heavy edits; with very short slang tokens, thresholds miss true matches (low recall) or admit too many spurious ones. For verlan in particular, its most basic behaviour is syllable

reversal, not local small substitutions, so these two alone are not adequate without further structure. Even using the Damerau-Levenshtein distance, which penalises an adjacent transposition as one step, verlan syllable reversal is not character substitution and is still inadequately modelled. Short strings triggering threshold sensitivity and false positives on fuzzy matching; see Navarro's survey for length-normalised threshold discussion. Cross-linguistic barriers and phonetic key collision rates are notorious, reviewed by Zobel and Dart. [50, 49, 51]

| **Method** | **Useful for** | **Limitations (for slang/verlan)** |
|---|---|---|
| Soundex | English-like names; minor spelling variants with similar pronunciation | Language-/name-centric; many key collisions (lower precision); poor on short tokens; inconsistent handling of diacritics; fails on syllable inversion. |
| Metaphone / Double Metaphone | Finer-grained phonetic keys; fewer collisions vs. Soundex | Still language-specific; limited coverage for French phonotactics; fails when phonology changes drastically (e.g., verlan). |
| Levenshtein distance | Typos, missing letters, noisy OCR; tunable similarity thresholds | Ignores pronunciation; syllable flips incur large distances (low recall); short tokens yield many false positives at practical thresholds. |
| Hybrid (phonetic + edit) | Balances recall/precision; practical for lexicon lookup | Depends on lexicon coverage; brittle for novel forms; does not infer unseen variants. |

Table 1: Classical fuzzy-match methods: strengths and limitations for slang/verlan.

Afterwards, scholars introduced additional algorithms and surveys: Philips's Metaphone (1990) and Double Metaphone (2000), both primarily designed for English — Double Metaphone adds some cross-language rules but is not intended for French phonotactics; Kukich's (1992) survey of spelling-correction techniques; Sproat et al.'s (2001) systematisation of Non-Standard Word (NSW) normalisation; and Aw et al.'s (2006) phrase-based MT for SMS normalisation. [24, 25, 26, 27, 28] While these are not directly slang-detection research, their methodology increasingly informed later slang-focused work.

### 2.2.2 2016-2019: Dictionary Search

The easiest way we can think of dealing with slang is to use a dictionary — just like how we look up a word that we do not know. The pros and cons are highly similar to consulting a dictionary. It is fast (if using a digital one) and accurate. On the other hand, because it is purely fixed data, it only works with existing words and thus cannot identify newly invented ones.

Examples of existing slang dictionaries include SlangNet, SlangSD, and SLANGZY[16, 17, 18]. As for French slang dictionaries, we have, for example, *Dictionnaire du chilleur*[19]. Specifically for verlan, the report identifies several online dictionaries, including *Dictionnaire Interactif du Verlan*[9], Wiktionary[10], and *Dictionnaire Verlan*[11].

With these dictionaries on hand, the reuse of a lookup-type verlan identifier is feasible. However, two factors prevent direct reuse: breakdowns in coverage and erratic entries, and the fact that most resources are community-composed, not formally edited — so accuracy, updating policies, and quality control vary. In addition, licensing on some sources may restrict redistribution. Our own dictionary, too, is researcher-compiled; to minimise risk of quality, we keep track of provenance and employ the gold/silver/bronze levels of quality (Table 2), and we use it primarily as a seed dictionary and rule-based baseline.

Although dictionaries have the drawbacks mentioned above, they remain essential resources for implementing LLM-based approaches, as discussed later. Consequently, new dictionaries continue to be produced.

### 2.2.3 Meanwhile, for Fuzzy Search

A few such example applications are Beaufort et al.'s French SMS normalisation hybrid finite-state approach, Han and Baldwin's English Twitter/SMS unsupervised lexical normalisation, and the W-NUT 2015 shared task on English Twitter normalisation [29, 30, 31].

Although not directly about slang detection, these studies were beneficial in three particular ways: (1) they provided operational pipelines for normalising noisy user-generated text (e.g., a hybrid finite-state rule/model archi-

[9] https://ecoleng.com/verlan-comprendre-argot-francais-parler/dictionnaire-interactif-du-verlan

[10] https://en.wiktionary.org/wiki/Category%3AVerlan

[11] https://zlang.fandom.com/fr/wiki/Dictionnaire_Verlan

tecture; cascaded detection→candidate-generation→context-sensitive selection); (2) they released datasets and/or standardised evaluation suites (Han & Baldwin's Twitter corpus; W-NUT's shared-task data and metrics); and (3) they reported where dictionary/rule-based and similarity/LM-based approaches tend to succeed or fail. In the cited work, Beaufort et al. reported WER $\approx$ 9.3% and BLEU $\approx$ 0.83 on French SMS, whereas Han & Baldwin showed that combining dictionary lookup, morpho-phonemic similarity, and context features attains state-of-the-art performance on Twitter/SMS (surpassing any single component), with clear gains from integrating lexical resources and context models.

Major constraints for our purposes include language- and domain-specific heuristics/lexicon coverage, reduced out-of-domain robustness (e.g., long-tail OOV in Twitter), and task setups that constrain normalisation to single-token targets rather than transformations appropriate for structural rearrangements such as syllable inversion—therefore, additional modelling is required to handle verlan.

#### 2.2.4 2020-2025: Fuzzy Search + Slang Corpus = BOOM

Since 2020, it has been progressively blended with crowd-sourced slang sources and neural models.

**Urban Dictionary embeddings (Wilson, 2020).** Wilson et al. trained 300-dimensional fastText vectors on ∼2M Urban Dictionary entries and evaluated them with simple fastText classifiers on sentiment and sarcasm benchmarks [32]. On Twitter sentiment, UD embeddings perform best among settings without additional word/character $n$-grams; when $n$-grams are added, GloVe-Twitter overtakes while UD remains competitive. On sarcasm detection, UD embeddings achieve the strongest accuracy among the compared embeddings. Limitations: the performance is domain-sensitive (strongest on informal/social-media text but weaker on formal/news-like text); the dictionary is noisy and English-centric; and the classifiers are shallow, so improvements may diminish with stronger encoders. [32]

**Slang or Not? (2024).** *Slang or Not?* constructs a sentence-level English dataset and compares classic ML baselines, CNN/BiLSTM models, transformer encoders, and LLMs [15]. The best supervised system reported is **BERT-large-uncased**, achieving accuracy = 0.87 (macro-F1 = 0.80; slang

F1 = 0.69, non-slang F1 = 0.92). Among LLMs, **GPT-4o-mini** attains accuracy = 0.86 [15]. Limitations noted by the authors include coverage gaps for rare or emerging slang, error clusters caused by "bad neighbours" (toxic or domain-specific context), proper nouns, and very long sentences; they also report underuse of LLM reasoning for marginal/ambiguous cases. [15]

**Toward Informal Language Processing (NAACL 2024 Findings).** The NAACL 2024 findings synthesise pitfalls and priorities for informal-language processing rather than reporting leaderboard-style scores [14]. Key issues highlighted are strong domain shift from formal training corpora to user-generated text, scarce/noisy or license-restricted resources, tokenisation and normalisation challenges for non-standard forms, and the need for carefully annotated, task-specific datasets. These observations motivate our decision to build a verlan-centred dictionary and sentence corpus and to evaluate detection with contextualised encoders. [14]

#### 2.2.5 Detecting Verlan?

The above results show a straightforward avenue for verlan detection. Fine-tuned transformer encoders previously outperformed classic ML and CNN/-BiLSTM at sentence-level slang detection (BERT-large-uncased: accuracy 0.87, macro-F1 0.80; slang F1 0.69) [15]; slang-aware embeddings also work on informal text but are domain-adaptive [32]. Building on such evidence, we employ contextualised transformer encoders as our supervised baselines and large LLMs as zero-shot reference points. We make this prediction for verlan in the following experiments. This progression, from fuzzy heuristics to contextual encoders, mirrors how language itself evolves — structured yet fluid, much like verlan.

# 3 Datasets

## 3.1 The Separated Structures

As of the time of writing, there are no published verlan datasets. Thus, this report has created two datasets: one is a lookup table mapping words in verlan to their standard French forms, named *GazetteerEntries* (hereafter *the dictionary*); the other contains example sentences for the words appearing

in the table, both in verlan and in standard French, with three entries per form, named *Sentences*. The general reasons for having two datasets are:

1. To separate rules and learning signals. The dictionary works as a lookup and a baseline for rule-matching: it provides word mappings, word variants, etc., whilst the sentences dataset is for detection and evaluation and illustrates *how* verlan appears in context. If mixed together, the model will not be able to distinguish tokens as dictionary knowledge or usage.

2. To improve reusability. The dictionary can be used independently on any corpus for rule-based verification, while the sentences dataset can be updated separately to add more community examples without modifying the dictionary, supporting modularisation of the pipeline.

3. For a cleaner evaluation. The dictionary can serve as a baseline while the sentences dataset can be split for training and testing, making results easier to interpret.

Generally speaking, the separation of the datasets can potentially make the model and the experiments clearer, explainable, and easy to extend. They could also contribute to LLM training and corpus creation in the future.

## 3.2 Visualisation of the Datasets

Figure 4 presents the attributes in the datasets and highlights how they relate to each other.

## 3.3 The Creations

As mentioned in Section 2.2.2, this report first checked and scraped sources that were available and had researcher-friendly copyright policies. Among those mentioned, *Dictionnaire Verlan* and Wiktionary contributed the most in terms of quantity. However, as they are not curated or officially published, their quality is not guaranteed. Moreover, many entries do not provide example sentences, which makes the creation of the sentences corpus harder.

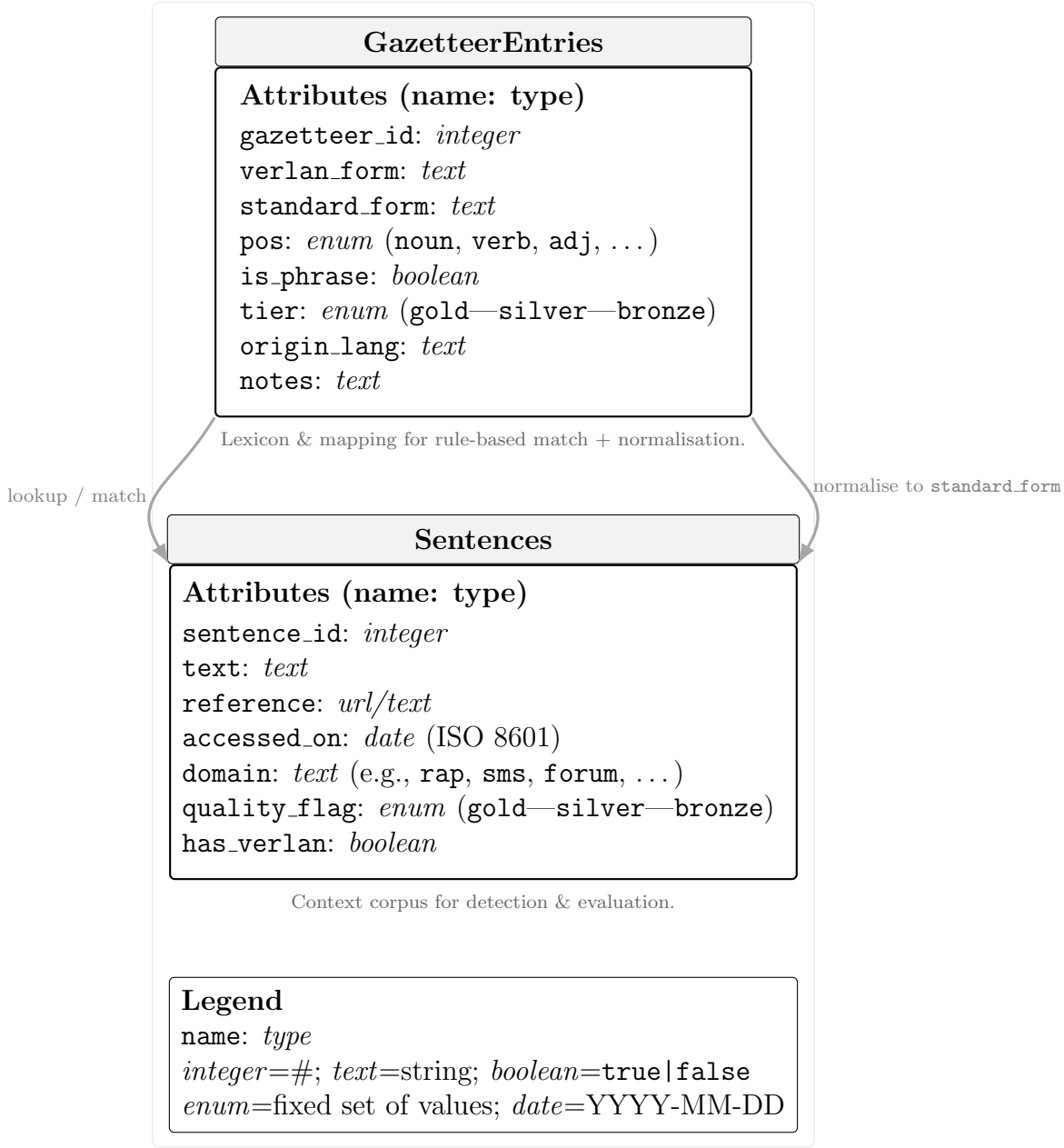



Figure 4: Overview of the GazetteerEntries lookup table and the Sentences corpus, including their key attributes.

### 3.3.1 Sampling

Although there is no clear estimate of the overall quantity of verlan, after searching, scraping, and combining, this report compiled a total of 1,086 verlan items, though some are merely spelling variants of the same word. For example, *foncédé* and *foncedé*[12] are counted separately as two entries; so are *keus* and *keuss*[13]. Notably, there are around 150 entries for which the report did not find their standard form; thus they have been categorised as *bronze* regarding their quality. To the best of our knowledge, the dictionary we have created contains the largest number of verlan entries among the public dictionaries we could find.

After creating the dictionary, the report searched for and scraped usage examples online to create the sentences corpus. The report also used Artificial Intelligence (AI) tools — specifically, OpenAI Deep Research[14] — for sentence scraping[15]. The results for sentences with a verifiable reference have been marked as *gold* quality; those without a verifiable reference have been marked as *silver* quality. For items for which the report could not find example sentences, we prompted ChatGPT-o3[16] to generate example sentences; these results have been marked as *bronze* quality. All results have been reviewed by the author of this report and are intended to undergo annotation in the future.

### 3.3.2 Balancing the Training Dataset

Because the sentence dataset will be used for LLM experiments, researchers have pointed out that an imbalanced dataset may affect the performance of trained LLMs. Therefore, balancing the number of sentences containing verlan and those not containing verlan becomes important[39].

To find an external, existing dataset that best balances ours, this report has summarised several traits of the current dataset:

1. Each entry is short, around 5 to 20 words.

2. Entries are relatively recent.

[12] Verlan of *défoncé*, often translated as *high (on drugs)* in English.
[13] Verlan of *sec*, translated as *dry* in English.
[14] https://openai.com/index/introducing-deep-research/
[15] Research has pointed out that this model gives better results in general[38].
[16] https://en.wikipedia.org/wiki/OpenAI_o3

3. Some entries are clips of longer sentences.

4. Entries include quotes and diverse annotation marks (e.g., !, ?, ...).

5. Entries are mostly informal and contain non-standard spellings other than verlan.

After consideration, this report has chosen to use the *title* column of the *Diverse French News* dataset published on HuggingFace[17], created by scholar Gustave Cortal[18]. It matches our dataset in terms of short entry length and was published in March 2022. Some entries even include quotations from celebrities. Although this dataset does not fully meet our requirement for diverse annotation marks, this report plans to apply pre-processing before tokenisation to trim off all annotation marks.

One small concern is that the news dataset mostly contains formal language, which may potentially affect the performance. While this might be the best available choice at present, this report will discuss this limitation in detail in future chapters.

### 3.3.3 Quality Tiers

To provide readers with a clearer understanding of the tier/quality schema introduced in this report, we have created a table for clarity:

Table 2: Quality tiers of the verlan datasets.

| Tier | Definition | Source |
|---|---|---|
| Gold | Verified with public reference | Public reference (URL/citation) |
| Silver | Plausible sentence without verifiable source | Scraped / semi-auto |
| Bronze | LLM-generated and manually reviewed | ChatGPT-o3 |

## 3.4 Final Dataset

At the time of writing, the datasets are not yet officially finalised. Although the structure of the two datasets is as shown in Section 3.2, in the dictionary dataset this report did not invest much effort in annotating the original language of the verlan items; the *note* column is also scarcely used. In

[17] https://huggingface.co/datasets/gustavecortal/diverse_french_news
[18] http://www.gustavecortal.com/

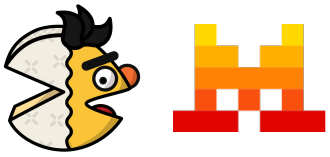
the sentences corpus, the *accessed_on* and *domain* columns are also scarcely annotated.

The main reason is that these columns were not used in the implementations. In fact, this report only used *gazetteer_id*, *verlan_form*, and *standard_form* in the dictionary, and *sentence_id*, *text*, and *has_verlan* in the sentences corpus. To us, the remaining columns are primarily for publishing the datasets and for potential advanced experiments in the future.

# 4 Building the Pipelines — Model Architectures and Specifications

## 4.1 Mistral 7B — Why?

The report has chosen Mistral 7B[19] developed by Mistral AI as the base model for the experimental design[34]. There are several reasons for this choice:



CamemBERT and Mistral AI Logos

1. Mistral AI is a French company[20]. The report argues that its training dataset is highly likely to contain more French slang contexts; therefore, its models may achieve better performance in French — especially in identifying verlan.

2. Mistral 7B is both powerful and relatively new. Verlan is a linguistic phenomenon that has been used daily online since the rise of social media. To identify verlan, we need up-to-date training datasets and models to keep pace with contemporary language use. While CamemBERT was also considered, it was released in 2019, whereas Mistral 7B was announced in September 2023, see Footnote [26][37]. Mistral 7B outperforms LLaMA 1 33B[21] and LLaMA 2 13B, two LLMs with larger parameter sizes[35, 36].

[19] https://mistral.ai/news/announcing-mistral-7b

[20] https://mistral.ai/about

[21] Although the original paper mentioned 34B, this report believes it was a typographical mistake. The evidence is that they referenced Meta's original paper, which did not include a 34B model.

3. Scholars may argue that the Mistral 8B model in les Ministraux[22] family would be a better choice, as it is the direct successor of the Mistral 7B model. However, the main reason we cannot use the newer model is that it is primarily designed for text generation rather than embedding. In fact, Mistral 8B does not support embeddings, unlike Mistral 7B — Salesforce AI[23] has published an embedding version[24] based on Mistral 7B.

4. Furthermore, to preserve full control and reproducibility, the report does not intend to use Application Programming Interface (API) calls, even though the Mistral Embed[25] model is available via API. We also do not intend to use proprietary models (e.g., ChatGPT, Grok[26]) as our main research models, for the same reasons stated above.

## 4.2 Zero-Shot Models

### 4.2.1 Mistral 7B Prompt Engineering with Vibe

Because we are experimenting with the performance of Mistral 7B (hereafter referred to as *Mistral*), designing a zero-shot test becomes important — foremostly, to determine whether Mistral has already learned how to identify verlan, and to what extent it can identify it correctly. Additionally, we can compare the performance of the zero-shot model with that of the trained models, to see to what extent our dataset augments the performance.

To achieve this, the report has designed the following simplified pipeline using prompt engineering:

[22] Literal meaning in English: the Ministrals.
[23] https://www.salesforce.com
[24] https://huggingface.co/Salesforce/SFR-Embedding-Mistral
[25] https://docs.mistral.ai/getting-started/models/models_overview/
[26] https://grok.com/

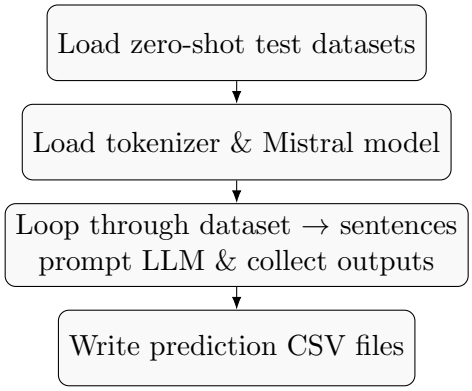



Figure 5: Zero-shot pipeline for Mistral

The prompt we used is as follows:

```
System: You are a linguist who identifies verlan
   (French reversed-syllable slang).
Reply with a single digit: '1' if the sentence
   contains verlan; otherwise reply '0'.
Do not include extra words.

User: Sentence:
{sentence}

Does this sentence contain verlan? Reply with one
    digit (0 or 1).
```

For each prompt, we start a new chat session to avoid the influence of the LLM's memorisation — reusing previous results may interfere with later performance.

A potential issue is that, from time to time, LLMs do not follow the system prompt and produce unexpected responses. The report has accounted for this — we use regular expressions to extract the numerical values mentioned in the response (i.e., 0 or 1) and store them in a separate column in the CSV file for easier post-processing. We also review the extracted labels manually to prevent inconsistencies or noise.

By doing so, the report believes that the accuracy and reliability of this pipeline are solid.

### 4.2.2 Zero-shot of the Most Powerful non deep reasoning LLM as reference

Considering that both the zero-shot and training experiments above are based on Mistral, the report aims to evaluate performance beyond the Mistral AI ecosystem. After reviewing the *Artificial Analysis Intelligence Index*[27], as shown in Figure 6, the report has chosen OpenAI's GPT-5 Codex (High)[28] model as the zero-shot reference. As the top-ranked model, it serves as a strong representative of the current state-of-the-art capabilities in verlan identification.

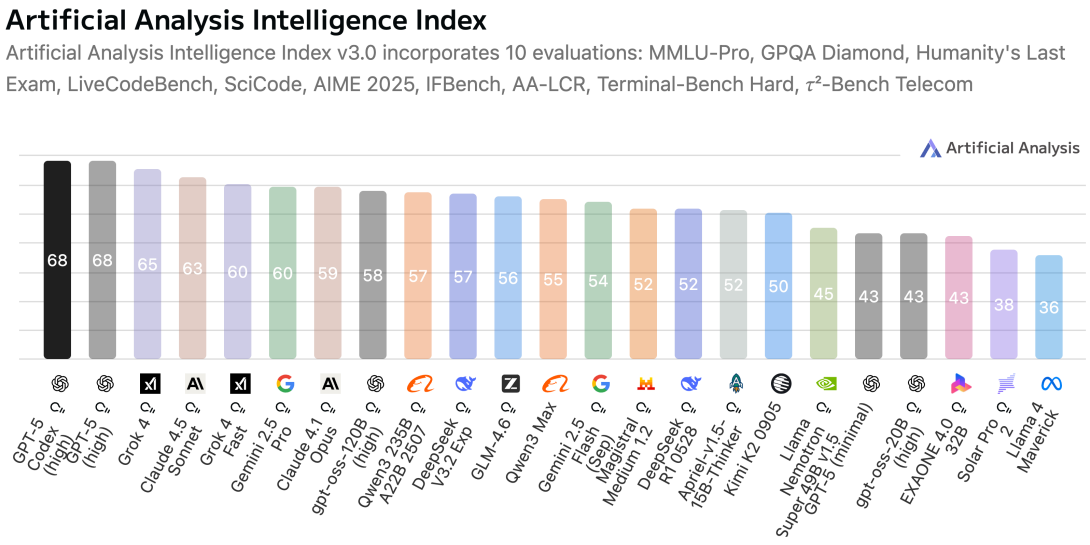



Figure 6: Leaderboard of the Artificial Analysis Intelligence Index (retrieved on 14 October 2025).

Following the principle of controlled experimental design, we used a prompt closely aligned with the one employed for Mistral:

```
[System message]
You are a linguist who identifies verlan (French
   reversed syllable slang). Ignore any prior
```

[27] https://artificialanalysis.ai/models/gpt-5-codex#artificial-analysis-intelligence-index

[28] https://openai.com/index/introducing-upgrades-to-codex/

```
   memories or cached context and follow only the
    instructions in this conversation. Do not
   browse the internet or use external tools;
   base your reasoning purely on the text you
   receive here. Reply with a single digit: "1"
   if the sentence contains verlan; otherwise
   reply "0". Do not include extra words,
   punctuation, or explanations.

[User message]
You will be given one or more French sentences.
   For each sentence, decide whether it contains
   verlan and answer with a single digit (0 or 1)
    per sentence, in the same order that the
   sentences appear.

Sentences to evaluate:
{sentences}
```

Notably, because GPT-5 Codex (High) is a reasoning-oriented model, its responses are typically slower, and it also has monthly usage limitations[29]. Therefore, instead of sending each sentence individually with the prompt, we chose to batch all sentences together in a single request. The maximum token size was taken into account, and the total length did not exceed the model's limit of approximately 400,000 tokens.

## 4.3 Training Models

This section introduces the methodology behind the training models. It first explains the pipeline from input to output and how the datasets are split and used within it. Then, it justifies the technical details of the specific hyperparameters and training platforms. Finally, it presents other pipelines that were considered but not implemented in this experiment, along with the rationale behind those decisions.

[29]The author of this report holds a ChatGPT Plus subscription.

### 4.3.1 The Pipelines

To avoid confusing readers, the report simplifies the flowcharts to highlight only the key components in this section. The complete flowcharts are provided in the appendix.

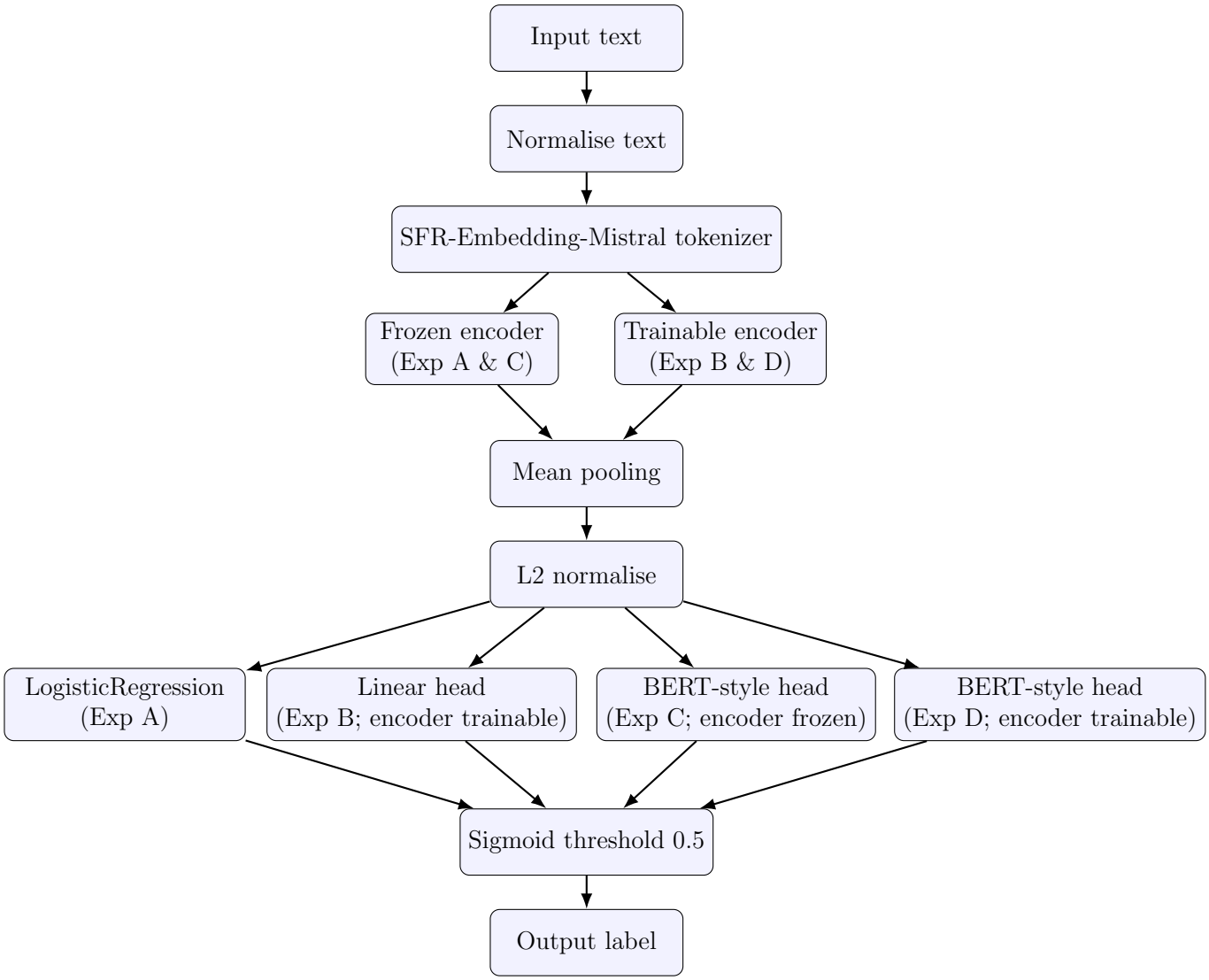



Figure 7: A compact view of the four verlan identification pipelines.
Exp A: Frozen Encoder + LogisticRegression Head
Exp B: End-to-End Encoder + Linear Head
Exp C: Frozen Encoder + BERT-Style Head
Exp D: End-to-End Encoder + BERT-Style Head

**Input** The input is the *Sentences* dataset we created. For editing and data management purposes, it is stored as an `.xlsx` table. It should be noted that the dataset was not converted into a special format before being fed into

the pipeline. Therefore, the labels indicating whether a sentence contains a verlan term remain in the input file when read by the program. However, we are confident that the sentence column was properly isolated, and that no visible data leakage occurred in the program code or during runtime.

### Normalise Text

**Why Not Preserve Upper Cases and Annotation Marks** As mentioned in Section 3.3.2, we have concerns that the diversity of annotation marks may affect the models' performance. Digging deeper, this is a good argument that even involves thinking about the role of verlan in a sentence from the LLM's perspective — a verlan might be an Out-Of-Vocabulary (OOV) word, or in other words, it might be treated as noise, like typographical mistakes. Indeed, scholars have pointed out that not only typographical mistakes but also annotation marks and the difference between upper and lower cases can all affect model performance[40].

Besides, during a smoke test, the report found that annotation marks and upper/lower case differences can indeed affect model performance. The model mislabels sentences that contain verlan as if they do not, when the sentence has not been normalised (i.e., trimmed off annotation marks and converted to lowercase). When it has been normalised, the model behaves normally and labels that sentence as containing verlan.[30]

This finding inspired the report to normalise the text before further experiments. Thus, we used regular expressions to trim off the annotation marks and convert the sentences to lowercase, while preserving the rest, including accented letters (e.g., é, à, ù).

**Tokenisation** Because all the encoders used in the four experiments originate from the same Mistral model, it is essential to ensure that the tokens received by the encoder match its expected input format. Therefore, we employ the tokenizer corresponding to Mistral, namely the one developed by

[30] Again, it was a non-official test run, so the report cannot claim that we have proven these changes would affect performance. But theoretically, it makes sense that if the LLM treats annotation marks, cases, and verlan all as noise, normalising the text would reduce unwanted noise and therefore increase accuracy. Further experiments can be conducted if needed.

Salesforce — *SFR-Embedding-Mistral* — to guarantee optimal model performance.

While it is indeed valuable to understand how this tokenizer segments each sentence, this tokenizer-encoder pair already represents the most appropriate configuration for our task. Hence, analysing its internal mechanisms in detail is considered unnecessary for this report.

**Encoder** As mentioned above, the encoders used in these experiments are identical, except that they are run in different modes: frozen or trainable. The reason for comparing a frozen encoder with a trainable one is, firstly, that we believe verlan identification is hypothesised to be a relatively straightforward task for a large language model (LLM).

Moreover, given that our dataset contains a limited number of entries, and considering that scholars have pointed out that in many NLP tasks, keeping most of the encoder frozen while fine-tuning only a few layers can still yield strong performance[41], a full fine-tuning approach may not be necessary.

Finally, taking into account both the time required and the expected outcome of partial fine-tuning to find the most optimal ratio between frozen and trainable layers, the report concludes that comparing the two bipolar cases — fully frozen and fully fine-tuned — is a more practical and representative approach.

**Calibrations** The report implements calibrations to the last hidden layer of the encoder, specifically, masking, mean pooling and L2 normalisation.

**Masking** The implementation of masking can be interpreted as below:

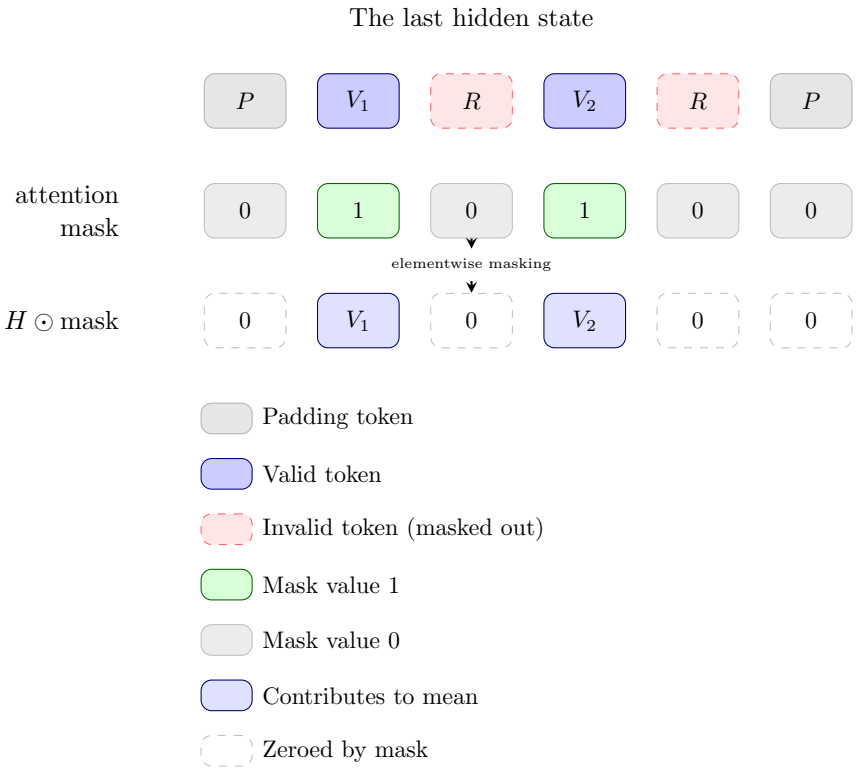



Figure 8: Visulisation of masking.

The last hidden state is the final layer of the encoder, where each token has its own hidden representation. It has the shape of a three-dimensional tensor, denoted as $\mathbf{H} \in \mathbb{R}^{B \times T \times D}$, where $B$ stands for the batch size (the number of sentences in a batch), $T$ stands for the sequence length (the number of tokens per sentence), and $D$ represents the hidden dimension (the size of each token vector).

Next, we apply the attention mask, which contains 1s and 0s. A value of 1 indicates a valid token, while 0 marks a padding or invalid token that should be ignored. However, this mask is *flat* — it is a two-dimensional tensor, $[B, T]$. To ensure it can be broadcast across all $D$ dimensions of each token vector, we add an extra dimension at the end of the tensor, resulting in a new shape of $[B, T, 1]$. This allows each 0/1 value to be applied consistently across the entire hidden vector of that token.

In the third line of the implementation, the mask is summed along the $T$ dimension, yielding the number of valid tokens in each sentence. The

resulting tensor has the shape $[B, 1]$.

It is always important to handle edge cases. If a hidden layer happens to be fully padded, the denominator in the subsequent division could become zero. Therefore, we clamp the minimum denominator to 1 to prevent division-by-zero errors.

**Mean Pooling** After obtaining the mask, we multiply it with the original hidden state $\mathbf{H}$. Here, $\mathbf{H} \in \mathbb{R}^{B \times T \times D}$, while the mask has the shape $[B, T, 1]$. By doing so, the mask's last dimension is broadcast to match the $D$ dimension. As a result, the $D$-dimensional vectors at valid positions remain unchanged, whereas those at masked (i.e., padding or invalid) positions become zero. This operation yields a state with the shape $[B, T, D]$ — the same as the original hidden state, but with masking applied.

Next, we collapse the $T$ dimension by summing all token vectors, resulting in a tensor of shape $[B, D]$. We then divide it by the number of valid tokens to obtain the mean representation of the valid tokens. The mathematical expression of mean pooling can be formulated as:

$$\text{pooled}[b, :] = \frac{\sum_{t=1}^{T} \text{mask}[b, t, 1] \cdot H[b, t, :]}{\max\left(1, \sum_{t=1}^{T} \text{mask}[b, t, 1]\right)} \quad \in \mathbb{R}^D \tag{2}$$

In short, mean pooling computes the average along the $T$ dimension. The resulting tensor contains one $D$-dimensional vector per sample in the batch, resulting in the overall shape $[B, D]$.

**L2 Normalisation** Since we obtain different vectors across the batch, each with potentially varying magnitudes,

$$\mathbf{h}_{\text{pooled}}[b, :] \in \mathbb{R}^D \tag{3}$$

their norms can differ. However, in the subsequent steps, we care more about the *direction* rather than the magnitude of these vectors. Therefore, we apply L2 normalisation to project all vectors onto the unit hypersphere, unifying their magnitudes to 1:

$$\|\mathbf{h}_{\text{norm}}[b, :]\|_2 = 1 \tag{4}$$

The normalised vectors are computed as:

$$\mathbf{h}_{\text{norm}}[b,:] = \frac{\mathbf{h}_{\text{pooled}}[b,:]}{\|\mathbf{h}_{\text{pooled}}[b,:]\|_2 + \epsilon} \quad \in \mathbb{R}^D \tag{5}$$

where the L2 norm is defined as:

$$\|\mathbf{h}_{\text{pooled}}[b,:]\|_2 = \sqrt{\sum_{i=1}^{D} (\mathbf{h}_{\text{pooled}}[b,i])^2}, \tag{6}$$

and $\epsilon$ is a small constant added to prevent division by zero.

By applying L2 normalisation, we ensure that the similarity between samples depends solely on the angular difference between their directions, which leads to more stable and comparable representations.

**The Classifiers — Logistic Regression or BERT** The primary difference among the four experiments lies here — not only in the choice between Logistic Regression and BERT, but also in the implementation framework, namely scikit-learn versus PyTorch.

**Logistic Regression with scikit-learn** For the experiment using a frozen encoder with a *Logistic Regression* head, we chose to employ scikit-learn (sklearn)[31]. It is simple to implement, and its `LogisticRegression` function internally handles both the loss computation and the optimiser. In contrast, the other three experiments involve learning and therefore use PyTorch[32] instead. Unlike scikit-learn, PyTorch does not provide built-in loss or optimiser functions for such cases, so these components are implemented explicitly, as illustrated in Figure 8. For the latter three experiments, both BERT and Logistic Regression (the linear layer) act as *heads*, whereas this experiment is referred to simply as *Logistic Regression*.

**BERT-Style Head** Regardless of whether it is Logistic Regression or BERT, both function as the classifier component — they process the normalised sentence vectors to determine whether a sentence contains verlan or not. Logistic Regression is merely a linear classifier; it cannot learn potential semantic patterns in the same way as the Mistral encoder. As mentioned

[31] https://scikit-learn.org
[32] https://pytorch.org/

earlier, CamemBERT serves as an alternative LLM to Mistral in this experiment. To combine the advantages of both and maximise their potential, we therefore employ BERT as another type of classifier.

However, BERT itself, much like the zero-shot Mistral tested earlier, is a complete LLM — it contains all the essential components such as a tokenizer, an encoder, and a classifier. Thus, it would be impractical to connect an entire BERT model after the Mistral encoder, as this would result in a redundant pipeline: Mistral embedder, Mistral tokenizer, Mistral encoder, BERT tokenizer, BERT encoder, BERT classifier, and so on. Therefore, we only utilise the *classifier head* from BERT.

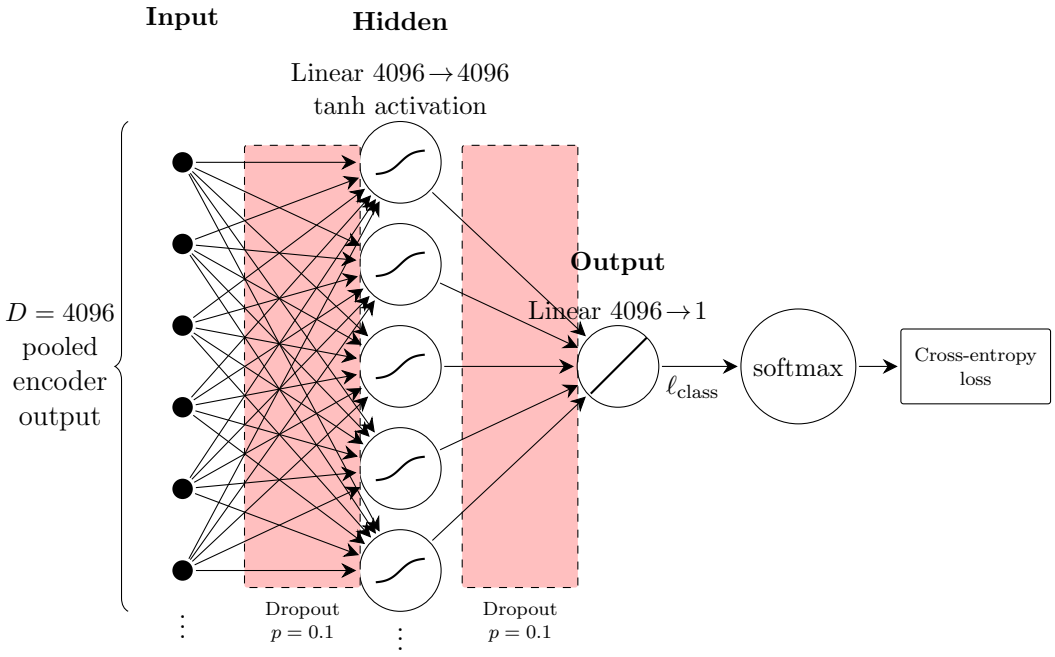



Figure 9: Classic BERT classification head.

Figure 10 illustrates the internal structure of a standard BERT classifier. It receives the pooled output from the encoder with a dimensionality of 4096, then applies dropout — randomly setting 10% of the neurons to zero to prevent overfitting. Since the neurons are linear, they are subsequently transformed with a `tanh` activation for improved non-linearity within the hidden layer. After that, dropout is applied again before mapping the 4096 hidden neurons to a single linear output neuron. Finally, a softmax function

converts the output into a probability distribution, which is then passed to the cross-entropy loss for evaluation.

However, because we are performing model fusion — that is, blending layers from two different LLMs — certain adaptations are required:

1. We have not only applied mean pooling but also normalisation to the output of the Mistral encoder. While the original BERT classifier uses only pooled features, we include normalisation to maximise fusion performance.

2. In the classic BERT classifier, the output logit is followed by a softmax function and a loss calibration step. In our case, we require a binary output (0 or 1), so we adopt a different loss function, calibration method, and thresholding strategy to produce binary results consistent with the design of the *Logistic Regression with scikit-learn* experiment.

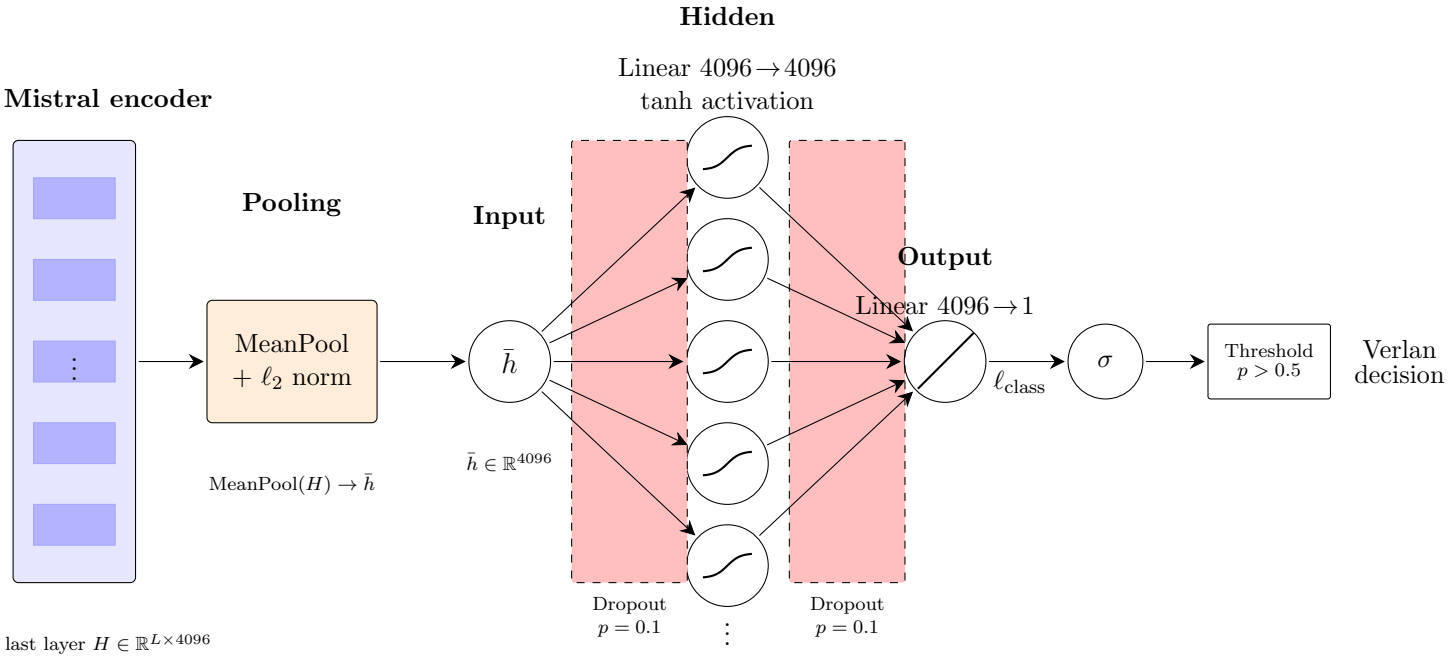



Figure 10: BERT-style detection head.

As shown in Figure 11, we modify the standard BERT classifier accordingly. We continue to refer to it as *BERT*, but use the term *BERT-style*

to emphasise that structural adjustments have been made for model fusion optimisation.

**The Different Loss Function and Calibration** As mentioned above, after the linear output neuron, we use a different loss function — `BCEWithLogitsLoss` — and a calibrator, AdamW[42, 43].

The *Binary Cross-Entropy (BCE)* loss is used in binary classification tasks. It measures the cross-entropy between the predicted probability distribution and the true label:

$$\mathcal{L}_{\text{BCE}}(y, \hat{y}) = -\big[y \cdot \log(\hat{y}) + (1 - y) \cdot \log(1 - \hat{y})\big] \tag{7}$$

where

$$y \in \{0, 1\}, \quad \hat{y} = \sigma(z) = \frac{1}{1 + e^{-z}} \tag{8}$$

According to the formula, the essence of BCE is to minimise the cross-entropy between the predicted distribution and the true distribution.

The `BCEWithLogitsLoss` function in PyTorch applies the sigmoid operation to the logits internally and then computes the BCE loss. Its numerically stable formulation is:

$$\mathcal{L}_{\text{BCEWithLogits}}(y, z) = \max(z, 0) - z \cdot y + \log\left(1 + e^{-|z|}\right) \tag{9}$$

This formulation prevents numerical underflow or overflow when the model becomes overconfident — that is, when the logit $z$ is very large or very small. In such cases, a direct computation of BCE might yield 0 or $\infty$, causing the training to crash or the gradient to become `NaN`. Hence, `BCEWithLogitsLoss` is more numerically stable than the pure BCE function.

*Adaptive Moment Estimation (Adam)* is a self-adaptive learning rate optimisation algorithm[44]. It elegantly integrates concepts from mathematics, physics, and computer science — it is based on the idea of momentum[33] and takes into account both the first-order moment (the mean of gradients) and the second-order moment (the uncentred variance of gradients).

For each iteration $t$, given the gradient $g_t = \nabla_\theta \mathcal{L}(\theta_t)$, the Adam update

[33] https://en.wikipedia.org/wiki/Momentum

rules are defined as follows:

$$\begin{aligned} m_t &= \beta_1\, m_{t-1} + (1-\beta_1)\, g_t \\ v_t &= \beta_2\, v_{t-1} + (1-\beta_2)\, g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \alpha\, \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \end{aligned} \tag{10}$$

where:

- $\beta_1, \beta_2$ are the exponential decay rates for the first and second moment estimates (commonly 0.9 and 0.999);
- $\epsilon$ is a small constant for numerical stability;
- $\alpha$ is the learning rate;
- $m_t$ is the first moment estimate;
- $v_t$ is the second moment estimate.

*Adam with Decoupled Weight Decay (AdamW)* further improves the optimisation process by decoupling the weight decay from the gradient update, thus correcting the regularisation deficiency in Adam[43]:

$$\begin{aligned} m_t &= \beta_1\, m_{t-1} + (1-\beta_1)\, g_t \\ v_t &= \beta_2\, v_{t-1} + (1-\beta_2)\, g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t}, \quad \hat{v}_t = \frac{v_t}{1-\beta_2^t} \\ \theta_{t+1} &= \theta_t - \alpha \left( \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} + \lambda \theta_t \right) \end{aligned} \tag{11}$$

where $\lambda$ is the weight decay coefficient. Unlike Adam, AdamW does not add the L2 regularisation term directly to the loss function; instead, it applies the decay explicitly to the weights. This modification ensures a consistent regularisation effect and often leads to better generalisation performance.

By applying these two techniques — a numerically stable loss function and a decoupled regularisation optimiser — the model is expected to achieve improved convergence stability and higher predictive accuracy.

**The Sigmoid Threshold** All the computations above yield a probability between 0 and 1, yet this is ultimately a classification task. Therefore, a sigmoid function is applied at the output layer. While alternative calibration methods could be explored instead of using a fixed threshold, for simplicity we employ a hard threshold of 0.5 to distinguish between the two classes. Further experiments may extend this approach by investigating adaptive or learned thresholding strategies.

### 4.3.2 The Usage of the Dataset

We randomly split the dataset into three subsets:

- Train — 72.25%
- Validation — 12.75%
- Test — 15%

The training set is used for the model to learn verlan patterns from the data, the validation set helps prevent overfitting and tune hyperparameters, and the test set evaluates the model's performance on verlan sentences that the model has not seen before. The reasons for adopting this particular split are as follows:

- The dataset is not large, so a relatively high proportion of verlan sentences is required for training.
- There are not many hyperparameters to tune, so a smaller portion of validation data is sufficient.
- To obtain a more stable evaluation result, the test set is made slightly larger than the validation set.

### 4.3.3 Environment and Hyperparameters

**Environment** Aoraki[34] is the research computing cluster at the University of Otago, Otākou Whakaihu Waka. All experiments presented in this report were conducted on Aoraki, specifically

[34] https://rtis.cspages.otago.ac.nz/research-computing/cluster/index.html#

using the same Nvidia L40 GPU. All models were implemented and fine-tuned under 4-bit quantisation to improve both efficiency and energy consumption. Further details of the environment configuration can be found on the GitHub page of this project.

**Hyperparameters**

**Seeds** We conducted 20 trials for each experiment to reduce bias. The same set of random seeds, ranging from 1 to 20, was used across all four experiments.

**Batch Size** We used a batch size of 32 for all experiments.

**Maximum Length** The maximum sequence length was set to 512 for all experiments.

**Quantisation** As mentioned above, the encoder was quantised to 4-bit NF4 with BF16 compute precision.

**Epochs** For the trainable encoders, training was performed for three epochs.

For detailed hyperparameter configurations, please refer to the GitHub page of this project.

# 5 Evaluations, Results, and Analyses

In this chapter, the report presents the evaluation techniques and the testing datasets that were created for this study. We then analyse the results both in general and in detail, followed by a discussion of the model's overall performance.

## 5.1 Evaluation Methodology

### 5.1.1 Embedding Space

To evaluate whether verlan tokens occupy distinct positions in the embedding space, we visualise the embeddings immediately after tokenisation (i.e.,

before training the encoder). After comparing Principal Component Analysis (PCA)[35], t-Distributed Stochastic Neighbor Embedding (t-SNE)[36], and Uniform Manifold Approximation and Projection (UMAP)[37], we found that UMAP generally produces the most effective visualisation[45, 46, 47].

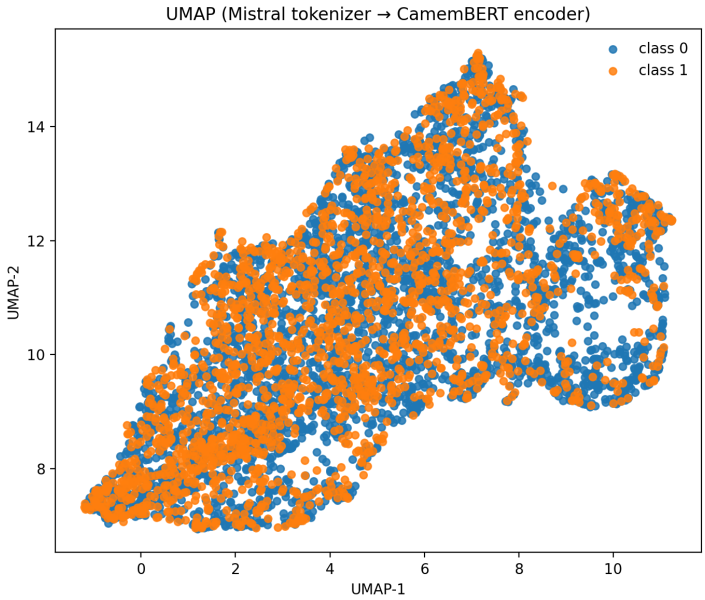



Figure 11: UMAP visualisations of the embedding space showing the distribution of verlan and standard French tokens
Class 1: verlan tokens
Class 0: normal tokens

Based on the results, the report cannot confidently conclude that there is a clear distinction between the positions of verlan tokens and other tokens in the embedding space. However, we strongly agree that traditional linear classification methods, such as Logistic Regression, are unlikely to perform well in this case, as the figure indicates a non-linear distribution pattern.

[35] https://en.wikipedia.org/wiki/Principal_component_analysis
[36] https://en.wikipedia.org/wiki/T-distributed_stochastic_neighbor_embedding
[37] https://umap-learn.readthedocs.io/en/latest/

### 5.1.2 Testing Datasets

To evaluate model performance, we created several testing datasets. These datasets are neither the sentence dataset nor the dictionary dataset that were used for training. We constructed three distinct datasets, each containing pairs of sentences: one version with verlan and the other with the corresponding verlan normalised into standard French.

1. **Daily Verlan** — 58 entries (29 pairs) of sentences that are frequently used by French speakers.
2. **Invented Verlan** — 50 entries (25 pairs) of sentences that were newly created to simulate the task of identifying novel verlan forms.
3. **Slang** — 50 entries (25 pairs) of sentences containing French slang and their normalised counterparts. All sentences in this dataset are labelled as not containing verlan.

The *Slang* testing dataset was included for the following reasons:

1. Slang can also be treated as a form of textual noise, much like verlan.
2. The model might learn to identify slang in general instead of verlan; therefore, this dataset allows us to verify whether such bias exists.

All sentences in these datasets do not appear in the training data.

Although the testing datasets are relatively small, this scale is sufficient for preliminary evaluation. However, the report considers this scale sufficient for evaluation purposes.

### 5.1.3 Testing Schema

**Zero-shot Models** For the zero-shot models, particularly GPT-5 Codex (High) and Mistral 7B, we evaluated them only on the testing datasets rather than the training dataset. As implied by their name, zero-shot experiments do not involve training; thus, applying them to the training set would be unnecessary.

**Number of Trials** As mentioned above, each model was run 20 times with 20 different random seeds to reduce bias.

**Storing the Results** For ease of analysis, the results of each sentence from each trial—across all four trained models and the zero-shot Mistral 7B—were stored in CSV format. For GPT-5 Codex (High), the results were stored in a spreadsheet for convenience.

### Analyse Methodology

**Confusion Matrix** In the analyses, a $2 \times 2$ binary confusion matrix is used. It also serves as the basis for computing both accuracy and the F1 score.

| **Total population** $= P + N$ | | **Predicted condition** | |
|---|---|---|---|
| **Actual condition** | **Positive (P)** | **True positive (TP)** | **False negative (FN)** |
| | **Negative (N)** | **False positive (FP)** | **True negative (TN)** |

Figure 12: Binary confusion matrix.

**Accuracy** To evaluate accuracy, we use the following formula:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{12}$$

**F1 Score** The F1 score is the harmonic mean of Precision and Recall, defined as:

$$\text{F1} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{13}$$

where

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}. \tag{14}$$

## 5.2 Results and Analyses

In this section, the report first presents the general result of the accuracy and the F1-score of the models. Then, the report discusses in separate cases with some interesting findings of the performance of the models. All the performance tests on the training models and the Mistral 7B zero-shot model have been conducted 20 times. The reference GPT model's performance test has been conducted once.

### 5.2.1 General F1-Score and Accuracy

Overall, all models produce positive detections — the accuracy across all models is above 50%.


Figure 13: A general comparison of the F1 score and accuracy across models.

For each individual model, before embedding and fine-tuning, Mistral 7B achieves an accuracy of around 69.4%, which is already considered high from the report's perspective. Training boosted accuracy by approximately 11.3, 7.3, 12.8, and 15.5 percentage points for the Frozen+LR, E2E+LR,

Frozen+BERT, and E2E+BERT configurations, respectively. The corresponding F1-score gains are around 9.9, 7.0, 12.1, and 14.6 points. These results demonstrate that training consistently improves model performance, even though the effect size is architecture-dependent.

The models that implement BERT as the classifier achieve higher accuracy and F1 scores than those using a Logistic Regression classifier. This result was anticipated, as the embedding space did not show a linear trend; hence, BERT performs better than Logistic Regression in capturing non-linear patterns.

Interestingly, the frozen models are not always the worst performers. The E2E+LR model — whose encoder was fine-tuned and uses a Logistic Regression head — achieves the lowest accuracy and exhibits the largest variance. The report suspects that this may be because fine-tuning with a limited dataset introduces additional noise, while the models with a BERT classifier can leverage their loss functions and optimisation mechanisms to mitigate this noise and achieve higher accuracy. Further experiments would be worthwhile to confirm this hypothesis.

Among the four trained models, the E2E+BERT model achieves the best overall performance. This demonstrates that model fusion between Mistral 7B and CamemBERT can be effective when implemented correctly, and that full fine-tuning with a non-linear classifier may be the optimal approach for verlan identification.

The reference zero-shot model, GPT-5 Codex (High), maintains the highest accuracy of 91.8% even without additional training, which is 22.4 percentage points higher than that of the zero-shot Mistral 7B. The F1 score demonstrates a similar trend. Although it is a proprietary model, these results allow us to observe how a top-tier model performs on this task and to appreciate the remarkable progress of the AI industry.

### 5.2.2 Common Verlan Performance by Model

The figure below illustrates the performance of the six models on the *Daily Verlan* testing set[38].

[38]The legend of the figure should be *common* verlan performance.

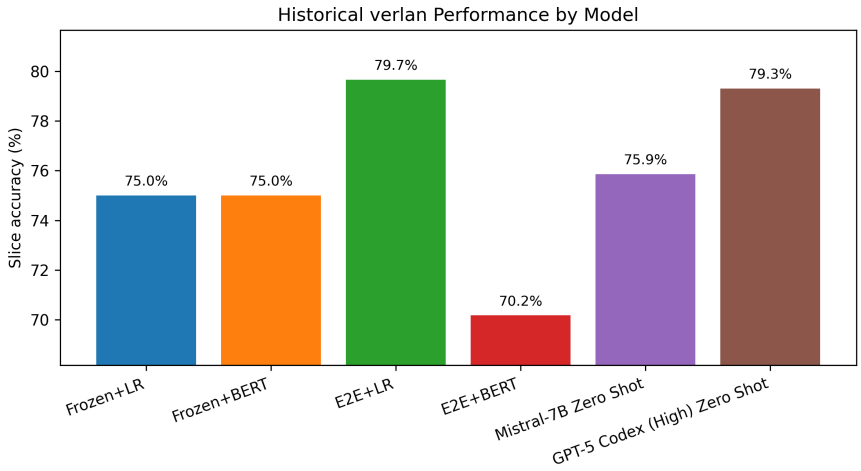



Figure 14: Models' performance in common verlan identification.

Interestingly, in this scenario, the results do not fully align with those presented in the previous section. Although the evaluation dataset is similar to the *Daily Verlan* testing set, it is not identical, and their outcomes are therefore expected to differ slightly. The report argues that this discrepancy arises because the evaluation set contains 853 entries, whereas the *Daily Verlan* testing set includes only 29 pairs. The smaller sample size naturally increases metric variability — but this does not merely imply higher noise. The testing set features shorter, high-frequency sentences, which tend to favour the frozen models. Indeed, the two frozen models achieve accuracy levels comparable to the zero-shot Mistral model.

For the fully fine-tuned models, we suggest that the evaluation split from the training dataset contains longer contexts, where these models recover precision. In contrast, within the testing set, the precision decreases due to shorter contexts, introducing additional noise and bias. Hence, although the E2E+LR model achieves a slightly higher accuracy than the zero-shot GPT-5 model, this difference is likely attributable to noise rather than genuine improvement.

When comparing with the E2E+BERT model, an opposite effect may be observed. The introduction of a trainable BERT classifier might have produced a contrary outcome compared to what was observed earlier. We hypothesise that because both the encoder and classifier are trainable in this case, when precision is lost, it degrades more significantly. Conversely, the

model with a trainable encoder and a non-trainable linear head may preserve more precision, as the untrainable head provides stability. Additionally, the loss function and optimisation strategy discussed in the previous chapter may have effectively smoothed out noise, allowing this configuration to outperform the two frozen-encoder models.

Again, this remains an assumption derived from scientific deduction. Further experiments would be valuable to validate these findings and strengthen the report's conclusions.

### 5.2.3 Invented Verlan Performance by Model

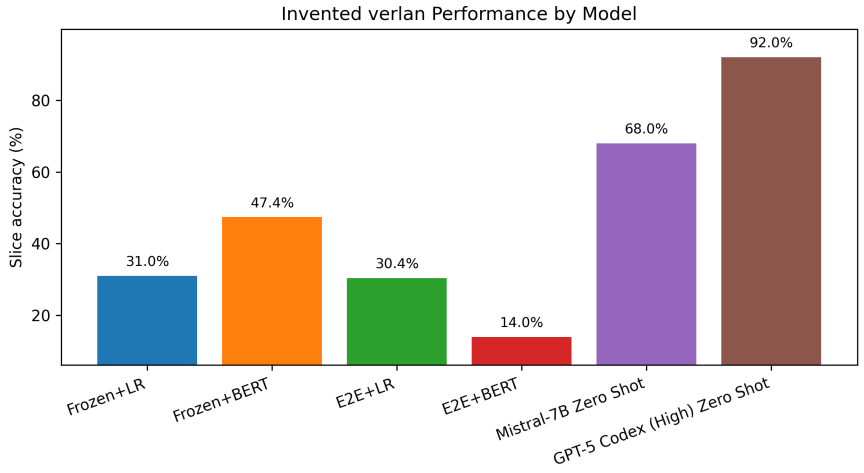



Figure 15: Invented verlan performance by model.

An interesting phenomenon is also observed here — the two zero-shot models outperform the four trained models on invented verlan recall by up to 78 percentage points (92% for GPT-5 Codex (High) versus 14% for E2E+BERT). In terms of accuracy, the gap remains substantial at roughly 42 percentage points. On average, there is a clear performance gap between the zero-shot models and the trained ones, with the trained models performing even worse than the zero-shot Mistral model.

The report would like to leave a hook here for readers to reflect on, before moving on to the next experiment.

### 5.2.4 Slang Controls Performance by Model

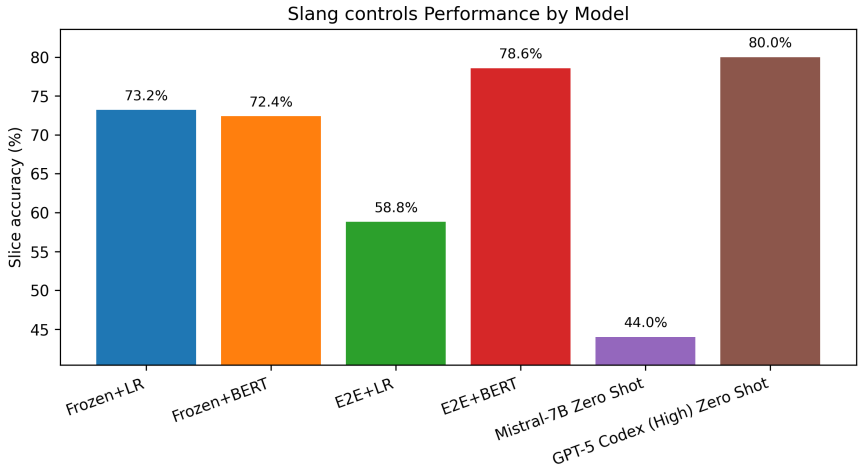



Figure 16: Slang control accuracy.

In this figure, everything seems to have flipped once again — the four trained models surpass their zero-shot counterparts by up to 34.6% in slang specificity. The report notes the low specificity (44%) of the Mistral zero-shot model, which may indicate that the model did not fully understand the task — specifically, what verlan is and its linguistic traits — thus performing close to random chance (50%).

After training, however, all four models appear to have resolved this issue. The two models with frozen encoders produce similar results, the E2E+LR model performs slightly below them, while the one with the BERT classifier performs the best among the trained models. This outcome closely resembles what we observed in the evaluation split dataset. The GPT-5 zero-shot model continues to perform as the overall best model.

**A Discussion on the Hook** Returning to the hook, the results of these two analyses seem to suggest something intriguing — the models may have learned to distinguish sentences that *do not* contain verlan, yet they may not have fully learned to identify sentences that *do*. Since this is a binary classification task, model performance could appear to improve simply by recognising what is *no* rather than what is *yes*. However, we argue that

in cases where "not no" is not equivalent to "yes — for instance, in a three-way classification setting — the model's performance would likely deteriorate significantly compared to this binary task.

### 5.2.5 And Yet Something Advanced

The sections above discuss only what we can observe on the surface. In this section, the report attempts something more advanced, aiming to explore the underlying characteristics of the models' performance in greater depth.

**Is E2E+BERT the Real King?** The figure below presents the specificity–recall distribution. The closer a point lies to the top-right corner, the better the model performs.

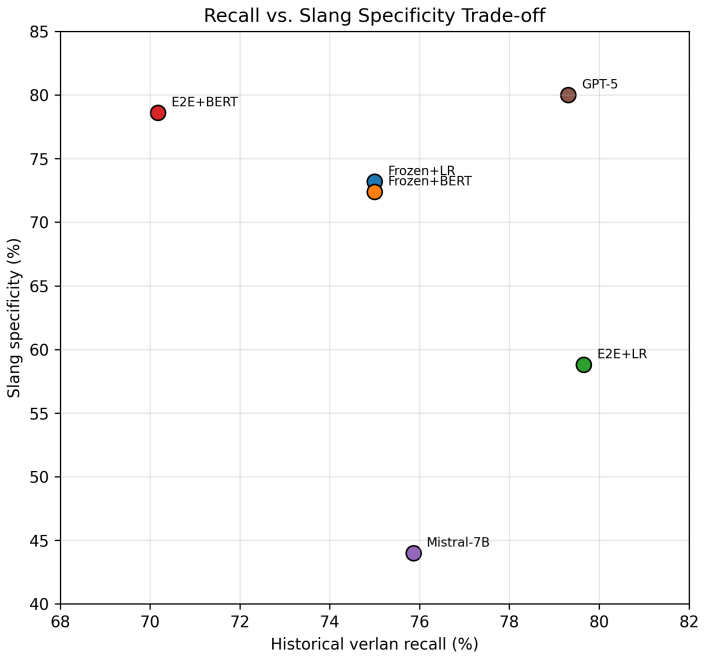



Figure 17: Recall–specificity trade-off across models.

Effectively, a model that achieves both high slang specificity and high daily verlan recall can be considered top-tier. Interestingly, according to this metric, GPT-5 Codex (High) performs the best, followed by the two models

with frozen encoders. This observation aligns with our earlier discussion: frozen embeddings often perform better on small datasets. In contrast, the two E2E models appear to be more biased.

**Small Dataset Caused More Instability?** The figure below shows the stability of the four trained models and includes the two zero-shot models for reference. The bars represent mean recall, with standard deviation error bars.

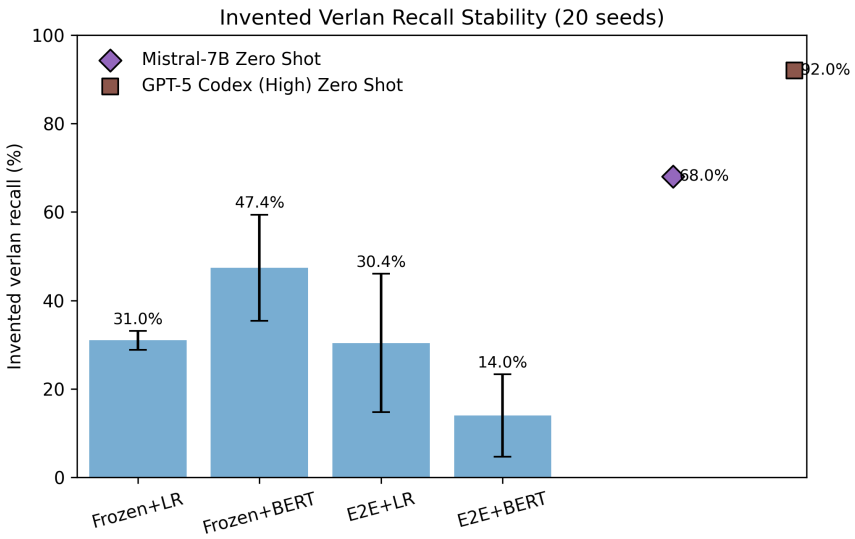



Figure 18: Invented verlan recall stability.

Judging from the standard deviation bars, the three models that include trainable components exhibit higher variance than the fully frozen model. Given that small datasets tend to yield better performance with frozen models, the report has reason to believe that making components trainable on a small dataset does lead to learning, but also introduces additional noise, resulting in higher standard deviation.

**Again: Is E2E+BERT the Real King?** The figure below depicts the link between slang false alarms and overall false positives.

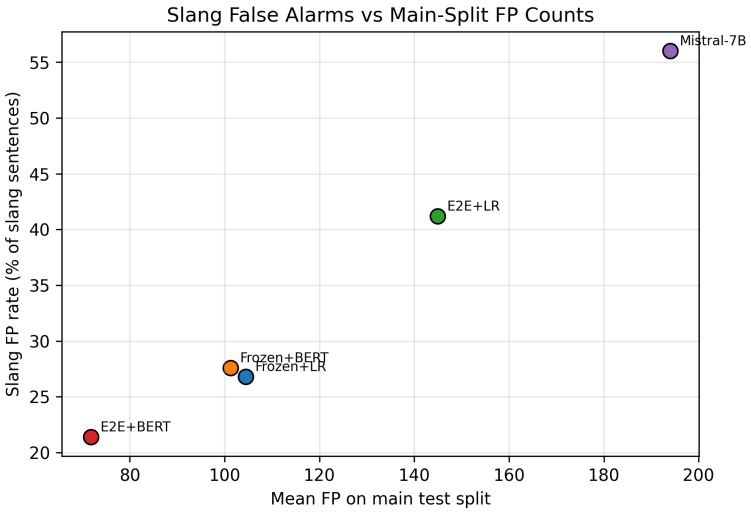



Figure 19: Link between slang false alarms and overall false positives.

The report has discovered that models which over-trigger on slang also tend to accumulate more false positives on the main test split. Conversely, models that trigger fewer slang false positives also show fewer false positives overall[39]. However, given that a $2 \times 2$ confusion matrix contains four attributes, this correlation does not necessarily imply that models closer to the bottom-left corner achieve more true positives — in other words, higher accuracy. Type I and Type II errors may still occur here.

## 5.3 Conclusion and Limitation

We acknowledge that this series of experiments has several limitations, and we encourage future researchers to explore this area further. In conclusion, the report finds that the proposed training dataset effectively improves the accuracy of verlan identification. In most cases, the model with a trainable encoder and a BERT-like classifier (E2E+BERT) outperforms the other trained models. In certain cases, however, the frozen encoder models achieve higher accuracy in identifying verlan sentences by selecting the correct ones. The zero-shot GPT-5 Codex (High) model remains the most stable and reliable across all scenarios.

[39]The GPT-5 model is not included, as the evaluation split dataset was not tested on it.

# 6 Discussion and Outlook

Just as the purpose of creating verlan was to conceal meaning behind words, Large Language Models (LLMs) tend to conceal themselves, turning into black boxes. So does our brain, and perhaps our universe. For now, scientific research is our only remedy — it forces us to think logically — yet we often regret: regret that, for the Mistral zero-shot case, the experiments were not run under different temperatures; regret that we did not find a better way to handle diverse spellings and declensions, as Russell and Levenshtein once did. We even regret hiding ourselves behind pronouns like "this report" and "we," when, in truth, we are always alone.

Yet we hope. We hope that more people will join us—not to suffer the same regret, but to break this eternal curse by seeking to understand the unknown — just as we strive to understand how LLMs learn Out-Of-Vocabulary (OOV) words such as verlan. And so, we wait.

# References


[1] Radjabov, Ruslan Rajabmurodovich. *Understanding "verlan" in the French Language*. Web of Scientist: International Scientific Research Journal, vol. 6, no. 3, 2025, pp. 368-372. Available at: https://webofjournals.com/index.php/3/article/view/3264.

[2] Bach, Xavier. *Tracing the origins of verlan in an early nineteenth century text*. Journal of French Language Studies, vol. 28, no. 1, 2018, pp. 1-18. Cambridge University Press. doi:10.1017/S0959269516000221.

[3] Olivier Sécardin. *Évolution du verlan, marqueur social et identitaire, comme reflet de la langue et de la société françaises*. Synergies Europe, no. 3, 2008, pp. 223-232. Available at: https://journal.lib.uoguelph.ca/index.php/synergies/article/download/1037/1859?inline=1.

[4] Rúa, Paula López. "Shortening Devices in Text Messaging." *Journal of Computer-Mediated Communication*, vol. 10, no. 4, July 2005. Wiley. doi:10.1111/j.1083-6101.2005.tb00268.x.

[5] Hajiyeva, Bulbul. "Translating Idioms and Slang: Problems, Strategies, and Cultural Implications." Acta Globalis Humanitatis et Linguarum, vol. 2, no. 2, 2025, pp. 284-293. doi:10.69760/aghel.025002123.

[6] DeepL. "DeepL Translator translates texts using artificial neural networks. These networks are trained on many millions of translated texts." *DeepL Blog*, 2020. Available at: https://www.deepl.com/en/blog/how-does-deepl-work.

[7] Wu, Yonghui, et al. "Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation." arXiv preprint arXiv:1609.08144, 2016. Available at: https://arxiv.org/abs/1609.08144.

[8] Michel, Paul, and Graham Neubig. "MTNT: A Testbed for Machine Translation of Noisy Text." *Proceedings of EMNLP*, 2018. Available at: https://aclanthology.org/D18-1050/.

[9] Zurbuchen, Lucas, and Rob Voigt. *A Computational Analysis and Exploration of Linguistic Borrowings in French Rap Lyrics*. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics — Student Research Workshop (ACL SRW 2024)*, 2024, pp. 200-208. DOI: 10.18653/v1/2024.acl-srw.27.

[10] Podhorná-Polická, Alena. *RapCor, Francophone Rap Songs Text Corpus*. In *Proceedings of the Fourteenth Workshop on Recent Advances in Slavonic Natural Language Processing (RASLAN 2020)*, 2020, pp. 95-102. Available at: https://nlp.fi.muni.cz/raslan/raslan20.pdf#page=95.

[11] Mekki, Jade; Lecorvé, Gwénolé; Battistelli, Delphine; Béchet, Nicolas. *TREMoLo-Tweets: A Multi-Label Corpus of French Tweets for Language Register Characterization*. In *Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021)*, Held Online, INCOMA Ltd., Sep 1-3, 2021, pp. 950-958. DOI: 10.26615/978-954-452-072-4_108.

[12] Panckhurst, Rachel; Lopez, Cédric; Roche, Mathieu. *A French text-message corpus: 88milSMS. Synthesis and usage*. Corpus [En ligne], 20 — 2020 (mis en ligne le 28 janvier 2020). DOI: 10.4000/corpus.4852.

[13] Pei, Zhengqi, Zhewei Sun, and Yang Xu. *Slang Detection and Identification*. In *Proceedings of the 23rd Conference on Computational Natural

Language Learning (CoNLL 2019)*, Hong Kong, China, 2019, pp. 881-889. Available at: https://aclanthology.org/K19-1082/.

[14] Sun, Zhewei, Qian Hu, et al. *Toward Informal Language Processing: Knowledge of Slang in Large Language Models*. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics (NAACL 2024)*, 2024. DOI: 10.18653/v1/2024.naacl-long.94.

[15] Anonymous. *Slang or Not? Exploring NLP Techniques for Slang Detection Using the SlangTrack Dataset*. ACL ARR (OpenReview) submission, December 2024 (ACL ARR 2024 December). Available at: https://openreview.net/forum?id=bISO3DD8sU.

[16] Dhuliawala, Shehzaad; Kanojia, Diptesh; Bhattacharyya, Pushpak. *SlangNet: A WordNet like Resource for Slang Words*. In: Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016). Portorož, Slovenia (2016). Available at: https://www.cse.iitb.ac.in/~pb/papers/lrec16-slangnet.pdf.

[17] Wu, Tianyang; Morstatter, Fred; Liu, Huan; et al. *SlangSD: Building, Expanding, and Using a Sentiment Dictionary of Slang Words for Short-Text Sentiment Classification*. Language Resources and Evaluation (2018). DOI: 10.1007/s10579-018-9416-0.

[18] Gupta, Vishal; Rani, Rekha; et al. *SLANGZY: A Slang Word Recognition System for Hindi-English Code-Mixed Social Media Text*. In: Proceedings of the 6th Workshop on South and Southeast Asian Natural Language Processing (WSSANLP 2019). Kolkata, India (2019). Available at: https://aclanthology.org/K19-1082.pdf.

[19] Parent, Philippe; Parent, André. *Dictionnaire du chilleur*. Éditions Somme toute (2024). ISBN: 9782925124351.

[20] Méla, Vivienne. *Le verlan ou le langage du miroir*. Langages, No. 101, Les javanais (Mars 1991), pp. 73–94. Published by Armand Colin. Available at: https://www.jstor.org/stable/23906698.

[21] Kaye, Jonathan D.; Lowenstamm, Jean. *De la syllabicité*. In: Dell, François; Hirst, Daniel; Vergnaud, Jean-Roger (eds.), *Forme sonore*

*du langage*. Hermann, Paris (1984), pp. 123–159. Available at: https://archive.org/details/formesonoredulangage.

[22] Russell, Robert C.; Odell, Margaret K. *Soundex system of indexing names*. U.S. Patent 1,261,167, filed June 3, 1918, and issued April 2, 1918. Available at: https://patents.google.com/patent/US1261167A/en.

[23] Levenshtein, Vladimir I. *Binary codes capable of correcting deletions, insertions, and reversals*. Soviet Physics Doklady, vol. 10, no. 8, 1966, pp. 707-710. Available at: https://nymity.ch/sybilhunting/pdf/Levenshtein1966a.pdf.

[24] Philips, Lawrence. *Hanging on the Metaphone*. Computer Language (1990). Available at: https://aspell.net/metaphone/.

[25] Philips, Lawrence. *The Double Metaphone Search Algorithm*. C/C++ Users Journal (June 2000). Available at: https://xlinux.nist.gov/dads/HTML/doubleMetaphone.html.

[26] Kukich, Karen. *Techniques for automatically correcting words in text*. ACM Computing Surveys (1992). DOI: 10.1145/146370.146380.

[27] Sproat, Richard; Black, Alan W.; Chen, Stanley; Kumar, Shankar; Ostendorf, Mari; Richards, Christopher. *Normalization of non-standard words*. Computer Speech & Language, 15(3):287–333 (2001). DOI: 10.1006/csla.2001.0169.

[28] Aw, AiTi; Zhang, Min; Xiao, Juan; Su, Jian. *A Phrase-Based Statistical Model for SMS Text Normalization*. Proceedings of the COLING/ACL 2006 Main Conference Poster Sessions (2006), pages 33–40. Available at: https://aclanthology.org/P06-2005/.

[29] Beaufort, Richard; Roekhaut, Sophie; Cougnon, Louise-Amélie; Fairon, Cédrick. *A Hybrid Rule/Model-Based Finite-State Framework for Normalizing SMS Messages*. Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics (ACL 2010). Available at: https://aclanthology.org/P10-1079.pdf.

[30] Han, Bo; Baldwin, Timothy. *Lexical Normalisation of Short Text Messages: Makn Sens a #twitter*. Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language

Technologies (ACL-HLT 2011), pages 368–378. Available at: https://aclanthology.org/P11-1038/

[31] Baldwin, Timothy; de Marneffe, Marie Catherine; Han, Bo; Kim, Young-Bum; Ritter, Alan; Xu, Wei. *Shared Tasks of the 2015 Workshop on Noisy User-generated Text: Twitter Lexical Normalization and Named Entity Recognition*. Proceedings of the Workshop on Noisy User-generated Text (W-NUT 2015). DOI: 10.18653/v1/W15-4319.

[32] Urban Dictionary Embeddings. *Urban Dictionary Embeddings for Slang NLP Applications*. LREC / ACL Anthology (2020). Available at: https://aclanthology.org/2020.lrec-1.586/.

[33] Sun, X.; et al. *Knowledge of Slang in Large Language Models*. Proceedings of the 2024 Annual Conference of the North American Chapter of the Association for Computational Linguistics (NAACL-Long 2024). Available at: https://aclanthology.org/2024.naacl-long.94/.

[34] Jiang, Albert Q.; Sablayrolles, Alexandre; Mensch, Arthur; Bamford, Chris; Chaplot, Devendra Singh; de las Casas, Diego; Bressand, Florian; Lengyel, Gianna; Lample, Guillaume; Saulnier, Lucile; Lavaud, Lélio Renard; Lachaux, Marie-Anne; Stock, Pierre; Le Scao, Teven; Lavril, Thibaut; Wang, Thomas; Lacroix, Timothée; El Sayed, William. *Mistral 7B*. DOI: 10.48550/arXiv.2310.06825

[35] Touvron, Hugo; Lavril, Thibaut; Izacard, Gautier; Martinet, Xavier; Lachaux, Marie-Anne; Lacroix, Timothée; Rozière, Baptiste; Goyal, Naman; Hambro, Eric; Azhar, Faisal; Rodríguez, Aurélien; Joulin, Armand; Grave, Edouard; Lample, Guillaume. *LLaMA: Open and Efficient Foundation Language Models*. DOI: 10.48550/arXiv.2302.13971

[36] Touvron, Hugo; Martin, Louis; Stone, Kevin; Albert, Peter; Almahairi, Amjad; Babaei, Yasmine; Bashlykov, Nikolay; Batra, Soumya; Bhargava, Prajjwal; Bhosale, Shruti; Bikel, Dan; Blecher, Lukas; Canton-Ferrer, Cristian; Chen, Moya; Cucurull, Guillem; Esiobu, David; Fernandes, Jude; Fu, Jeremy; Fu, Wenyin; Fuller, Brian; Gao, Cynthia; Goswami, Vedanuj; Goyal, Naman; Hartshorn, Anthony; Hosseini, Saghar; Hou, Rui; Inan, Hakan; Kardas, Marcin; Kerkez, Viktor; Khabsa, Madian; Kloumann, Isabel; Korenev, Artem; Koura, Punit; Lachaux, Marie-Anne; Lavril, Thibaut; Lee, Jenya; Liskovich, Diana; Lu, Yinghai; Mao, Yuning;

Martinet, Xavier; Mihaylov, Todor; Mishra, Pushkar; Molybog, Igor; Nie, Yixin; Poulton, Andrew; Reizenstein, Jeremy; Rungta, Rashi; Saladi, Kalyan; Schelten, Alan; Silva, Ruan; Smith, Eric Michael; Subramanian, Ranjan; Tan, Xiaoqing Ellen; Tang, Binh; Taylor, Ross; Williams, Adina; Xiang, Jian; Xu, Puxin; Yan, Zheng; Zarov, Iliyan; Zhang, Yuchen; Fan, Angela; Kambadur, Melanie; Narang, Sharan; Rodríguez, Aurélien; Stojnić, Robert; Edunov, Sergey; Scialom, Thomas. *Llama 2: Open Foundation and Fine-Tuned Chat Models*. DOI: 10.48550/arXiv.2307.09288

[37] Martin, Louis; Muller, Benjamin; Ortiz Suárez, Pedro Javier; Dupont, Yoann; Romary, Laurent; Villemonte de la Clergerie, Éric; Seddah, Djamé; Sagot, Benoît. *CamemBERT: a Tasty French Language Model*. DOI: 10.48550/arXiv.1911.03894

[38] Du, Mingxuan; Xu, Benfeng; Zhu, Chiwei; Wang, Xiaorui; Mao, Zhendong. *DeepResearch Bench: A Comprehensive Benchmark for Deep Research Agents*. DOI: 10.48550/arXiv.2506.11763

[39] Dong, Yuhui; Geng, Xin; Zhang, Jiayi; Song, Yue; Li, Xixin. *Understanding the Effects of Language-specific Class Imbalance*. DOI: 10.48550/arXiv.2402.13016.

[40] Al Sharou, Khaled; Li, Zheng; Specia, Lucia. *Towards a Better Understanding of Noise in Natural Language Processing*. Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021). DOI: 10.26615/978-954-452-072-4_007

[41] Lodha, Dhruv; Chen, Chiyu; Socher, Richard; Xiong, Caiming. *On Surgical Fine-tuning for Language Encoders*. Findings of the Association for Computational Linguistics: EMNLP 2023. DOI: 10.18653/v1/2023.findings-emnlp.204

[42] Paszke, Adam; Gross, Sam; Massa, Francisco; Lerer, Adam; Bradbury, James; Chanan, Gregory; Killeen, Trevor; Lin, Zeming; Gimelshein, Natalia; Antiga, Luca; Desmaison, Alban; Köpf, Andreas; Yang, Edward; DeVito, Zachary; Raison, Martin; Tejani, Alykhan; Chilamkurthy, Sasank; Steiner, Benoit; Fang, Lu; Bai, Junjie; Chintala, Soumith. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. NeurIPS 2019. DOI: 10.48550/arXiv.1912.01703

[43] Loshchilov, Ilya; Hutter, Frank. *Decoupled Weight Decay Regularization.* ICLR 2019. DOI: 10.48550/arXiv.1711.05101

[44] Kingma, Diederik P.; Ba, Jimmy. *Adam: A Method for Stochastic Optimization.* ICLR 2015. DOI: 10.48550/arXiv.1412.6980

[45] Pearson, Karl. *On Lines and Planes of Closest Fit to Systems of Points in Space.* Philosophical Magazine, 1901. DOI: 10.1080/14786440109462720

[46] van der Maaten, Laurens; Hinton, Geoffrey. *Visualizing Data using t-SNE.* Journal of Machine Learning Research, 2008. DOI: 10.48550/arXiv.1308.0719

[47] McInnes, Leland; Healy, John; Melville, James. *UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction.* arXiv preprint, 2018. DOI: 10.48550/arXiv.1802.03426

[48] Wagner, Robert A.; Fischer, Michael J.
*The String-to-String Correction Problem.*
Journal of the ACM, vol. 21, no. 1, 1974, pp. 168–173.

[49] Navarro, Gonzalo.
*A Guided Tour to Approximate String Matching.*
ACM Computing Surveys, vol. 33, no. 1, 2001, pp. 31–88.

[50] Damerau, Frederick J.
*A Technique for Computer Detection and Correction of Spelling Errors.*
Communications of the ACM, vol. 7, no. 3, 1964, pp. 171–176.

[51] Zobel, Justin; Dart, Philip.
*Phonetic String Matching: Lessons from Information Retrieval.*
Proceedings of the 19th Australasian Computer Science Conference (ACSC 1996), pp. 331–340.

# Appendix A Expanded Evaluation Artefacts

| Model | Test Acc | Test F1 | Test FP | Test FN | Slang Acc | Slang FP | Verlan Acc | Invente |
|---|---|---|---|---|---|---|---|---|
| Frozen+LR | $80.7\% \pm 0.9$ | $75.9\% \pm 1.3$ | 104.5 | 60.2 | 80.6% | 9.7 | 85.2% | 65.2 |
| Frozen+BERT | $82.2\% \pm 1.8$ | $78.0\% \pm 1.8$ | 101.3 | 50.5 | 81.8% | 9.1 | 85.4% | 70.8 |
| E2E+LR | $76.7\% \pm 7.1$ | $72.9\% \pm 7.5$ | 144.9 | 54.1 | 62.6% | 18.7 | 74.5% | 54.7 |
| E2E+BERT | $84.9\% \pm 4.9$ | $80.5\% \pm 5.1$ | 71.8 | 56.7 | 81.1% | 9.4 | 77.0% | 53.7 |

Table 3: Hold-out test aggregates (20 seeds) for trained detectors. Percentages report mean ± standard deviation across seeds; counts are means.

| Model | Historical recall | Invented recall | Slang specificity |
|---|---|---|---|
| Frozen+LR | $75.0\% \pm 3.3$ | $31.0\% \pm 2.2$ | $73.2\% \pm 2.3$ |
| Frozen+BERT | $75.0\% \pm 6.7$ | $47.4\% \pm 12.3$ | $72.4\% \pm 11.5$ |
| E2E+LR | $79.7\% \pm 6.4$ | $30.4\% \pm 16.1$ | $58.8\% \pm 13.0$ |
| E2E+BERT | $70.2\% \pm 5.3$ | $14.0\% \pm 9.6$ | $78.6\% \pm 7.3$ |

Table 4: Targeted-slice comparison across 20 seeds. Historical and invented columns are verlan recalls; slang column measures rejection accuracy on contemporary slang controls.

| Model | Historical recall | Invented recall | Slang specificity |
|---|---|---|---|
| Mistral-7B Zero Shot | 75.9% | 68.0% | 44.0% |
| GPT-5 Codex (High) Zero Shot | 79.3% | 92.0% | 80.0% |

Table 5: Zero-shot targeted breakdowns (single pass). Metrics computed on 29 historical verlan, 25 invented verlan, and 25 slang sentences respectively.

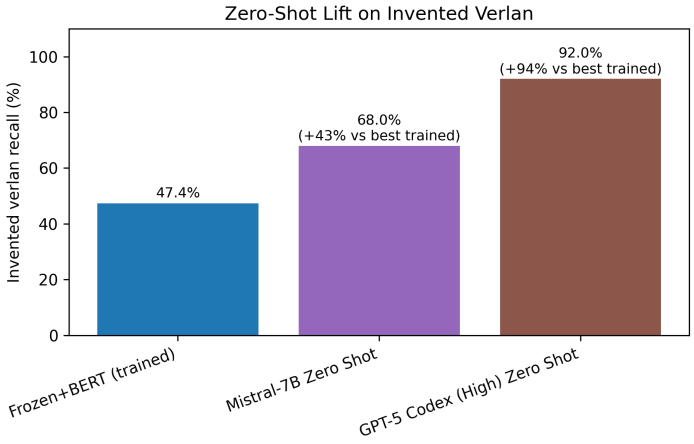



Figure 20: Zero-shot recall lift on invented verlan relative to the best trained detector (Frozen+BERT).