A Pipeline for Automated Dictionary Creation with Optional Human Intervention

Thomas Widmann

No Institute Given

Abstract

This paper presents a modular pipeline for automated dictionary creation using large language models (LLMs). It addresses the well-known limitations of prompting systems such as ChatGPT to produce entire entries in a single step – outputs that may read fluently but often lack structural consistency, transparency, originality and verifiability. The proposed system overcomes these weaknesses by decomposing the lexicographic process into a sequence of narrowly constrained, XML-validated stages, each guided by custom-crafted prompts and Document Type Definitions (DTDs). Rather than asking an LLM to "write a dictionary entry," the system treats it as a disciplined assistant performing a defined subtask under strict supervision.

At each stage – ranging from extracting and shortening corpus examples to grouping, defining, translating and formatting – the output is verified against an XML grammar and preserved for audit. This structure enforces reproducibility and allows human intervention at any point, combining the speed and adaptability of machine generation with the oversight and accountability of traditional lexicography. The process is entirely corpus-grounded: every example can be traced to a verifiable source, and every decision in the pipeline is documented. Errors can be corrected where they occur rather than through repeated prompting, and edited intermediate files can be reintegrated seamlessly into the workflow.

Technically, the pipeline is implemented in Python and designed to integrate easily with standard dictionary environments such as IDM's DPS system. It is language-agnostic and domain-independent: prompt files and DTDs can be adapted to any language pair, dictionary type or corpus source. The modular architecture also enables the insertion of new stages – for example, automatic tagging of usage labels, collocations or etymological notes – without altering the underlying structure. The system produces both machine-readable XML output and human-friendly Markdown files for editorial review, ensuring compatibility with established lexicographic and publishing workflows.

Two sample entries for the Danish adjective *nørdet* demonstrate that the pipeline achieves consistent formatting, transparent sourcing and idiomatic translations while avoiding plagiarism and hallucination. Evaluation suggests that each complete run (typically five stages) produces a usable draft entry at minimal cost and within seconds. The approach therefore provides a sustainable framework for dictionary production, especially for under-resourced languages or specialised terminologies where editorial time and funding are limited.

By embedding formal validation and corpus traceability into every step, the system offers a practical model for responsible integration of LLMs into lexicography. It shifts the human role from mechanical compilation to high-level editorial judgement, enabling lexicographers to supervise, refine and extend AI-generated content with full transparency. Released as open source under the MIT Licence, the pipeline invites adaptation, experimentation and community collaboration.

Keywords: dictionary creation; large language models; lexicographic automation; XML validation

1. Introduction

Automatic dictionary creation is not a new ambition. Lexicographers have long explored ways to streamline the compilation process, and tools such as Sketch Engine have introduced varying degrees of automation into lexicographic workflows.

The recent emergence of large language models (LLMs) such as ChatGPT has prompted renewed excitement, owing to their fluency, contextual awareness and surface-level coherence, at least since the recorded talk by de Schryver & Joffe (2023) was released. However, asking an LLM to generate a dictionary entry directly typically results in content that is inconsistent, unverifiable and, in some cases, plagiarised.

Our project builds on the insight that LLMs perform best when assigned narrowly defined, constrained tasks. Rather than treating ChatGPT as a fully fledged lexicographer, we treat it as a trainee working under strict supervision. It completes each small step according to explicit instructions, with all output validated against a formal schema and traceable to corpus data.

This paper introduces a fully implemented system for GPT-assisted dictionary creation. It is designed to produce entries that are internally consistent, corpus-grounded, structurally validated and editable using standard dictionary tools. We present the architecture, motivation and an early evaluation of the system, with all code and prompts released as open source (Widmann, 2025a).

2. Motivation

As highlighted by several authors, including Nichols (2023), Rundell (2023) and Widmann (2025b), prompting a large language model to produce a dictionary entry in a single step introduces several well-documented problems:

1. **Consistency**: The generated entries often lack a coherent structure, making them difficult to compare or integrate into larger lexicographic projects.
2. **Transparency**: There is no clear link between examples and real-world usage. They may appear plausible but are not traceable to verifiable sources such as corpora.
3. **Hallucination**: Some content is simply invented – for instance, inaccurate transcriptions, fabricated senses or fictional usage patterns – with no systematic way to detect or correct the errors.
4. **Originality**: Many generated definitions, particularly in English, appear to replicate existing dictionary entries, presumably drawn from the model's training data (cf. Nichols (2023)).
5. **Editability**: Output is rarely structured for direct import into dictionary-editing environments, limiting its practical usefulness.

The present system addresses each of these problems by enforcing structure, sourcing examples from corpora, validating against formal grammars and preserving every intermediate step for inspection and revision. This is, of course, an area attracting considerable

interest, but we believe the solution proposed here is novel – not LLM integration per se, which is already implemented in TLex and other tools, but the structured, phased approach adopted here.

These challenges are not unique to lexicography. Similar limitations have been observed in other fields where LLMs are asked to generate structured, verifiable content without guidance. What distinguishes the lexicographic case is the availability of large, curated corpora and formal entry templates, which make it particularly amenable to a stepwise, schema-driven solution. The approach outlined in this paper builds directly on these strengths.

3. Producing a Dictionary Entry Automatically

3.1 Preparing a Corpus Concordance

The system operates on keyword-in-context (KWIC) concordances provided as plain-text files. Each concordance entry should be separated by a double line break, and each line should represent an authentic usage example of the headword or search term. The system itself is source-agnostic: any tool capable of producing suitably formatted KWIC lines can serve as a source.

A small utility script, `kwic.py`, included in the GitHub repository, illustrates the basic idea. It scans a plain-text corpus and returns matches with a configurable amount of context on either side. This script is intended solely as an example and is not suitable for production use.[1]

The examples presented in this article were extracted from the Danish Language Council's internal 7-billion-word corpus, which primarily consists of Danish newspaper text from the past fifteen years. Users are encouraged to substitute their own corpus infrastructure and sources according to language, domain or project-specific requirements.

The system supports arbitrary corpora,[2] preprocessing methods and language pairs. As long as a KWIC file in the expected format is provided, the workflow remains fully compatible. However, the corresponding `.mdx` prompt file must be tailored to match the language and content of the input file.

3.2 The Prompt File

Each stage of the pipeline is controlled by a dedicated prompt, defined in a structured text file with the `.mdx` extension. These files consist of a sequence of named blocks, each comprising:

- a `#name` identifier (e.g. `extract_examples`),
- a `#prompt` section containing detailed instructions to the language model,

[1] If the reader does not have access to a corpus, they may use WikiExtractor (Attardi, 2015) to extract plain text from a Wikipedia dump. Further information can be found in the `README.md` file in the GitHub repository.

[2] Of course, the quality of the corpus feeds directly into the resulting dictionary entry.

- a `#example` illustrating the expected structure and content of the output in XML format,
- a `#DTD` defining the XML grammar that the output must conform to.

Different prompt files may be created for different language pairs, domains or dictionary types. In this document, we refer solely to `danish-english.mdx` (included in the GitHub repository). This file was constructed manually through trial and error, and can undoubtedly be improved. An `.mdx` file designed for a definition dictionary or for a thesaurus would consist of very different stages and prompts.

The prompts should also be adapted to the specific LLM in use. We currently employ the GPT-4o model, which yields good results, but different models may vary in how closely they follow instructions and how readily they improvise.

Each prompt stage in the system is defined as a block within a single `.mdx` file. Each block includes an instruction, a minimal XML example and a DTD specification. This modular setup ensures that the LLM is guided by clearly defined expectations at every stage.

Prompt stages can be added, modified or reordered to support different languages, translation directions or dictionary types.

3.3 Running and Rerunning the Program

The prompt chaining replicates traditional lexicographic logic (shorten → group → define → translate → label), offering a bridge between human and machine workflows.

At runtime, the `lexicographer.py` program reads the appropriate `.mdx` file and submits the prompts sequentially – together with the latest intermediate XML file – to the OpenAI API (by default using the GPT-4o model, though this can easily be changed). The returned output is validated against the DTD using Python's `lxml` library. If validation fails, the prompt is automatically reissued until well-formed and valid XML is produced. It is therefore essential that the DTD matches both the prompt and the XML example; otherwise, the validation step may continue failing indefinitely.[3]

Intermediate XML files are preserved and may be manually edited if required. The process can be resumed from any stage, ensuring that output is structured, auditable and readily adapted. This flexibility makes it possible to fix errors at the point where they occur – for instance, when a corpus example is shortened incorrectly or a translation is unsatisfactory – rather than rerunning the entire process and simply hoping for a better result. If the LLM struggles with a particular example, it may well encounter the same difficulty on subsequent runs. Without the ability to intervene mid-pipeline, users could be forced to repeat the full process multiple times. Direct correction is both more efficient and more predictable.

3.3.1 Example Prompt: Step 1 (*extract_examples*)

The following is a sample prompt used to extract examples from a corpus concordance:

[3] The `lexicographer.py` program does not retry indefinitely; by default, it attempts five times before giving up.

```
Here's a corpus concordance for the Danish lemma '$hw' ($pos). Please
skip any lines where the part of speech is something other than '$pos',
or where the word is part of a name or title. Then shorten each
remaining line to a single short sentence or sentence fragment in Danish
that includes the headword, making as few changes as possible.

The result must be a truthful representation of the original quotation,
meaningful in its own right, and useful as a dictionary example
(ideally 8-12 words long). To reiterate: all included lines must involve
the lemma '$hw' used as a '$pos'.

Here is a minimal example of the kind of XML I'd like you to produce:

<entry>
  <hw>ko</hw>
  <trgrp>
    <orig>træer og et rigtigt bondehegn. Faktisk manglede der kun en ko
          bag hegnet for at skabe en perfekt filmkulisse. Nogle vil</orig>
    <ex>Faktisk manglede der kun en ko bag hegnet.</ex>
  </trgrp>
</entry>

The XML must adhere to this DTD:

<!ELEMENT entry (hw,trgrp+)>
<!ELEMENT hw (#PCDATA)>
<!ELEMENT trgrp (orig, ex)>
<!ELEMENT orig (#PCDATA)>
<!ELEMENT ex (#PCDATA)>
```

Each stage produces output that becomes the input to the next. Validation is enforced using `lxml.etree`, and any failed stages are reattempted automatically.

3.4 Subsequent Stages of the Pipeline

While the first stage (`extract_examples`) focuses on extracting concise, corpus-based examples, the system continues through several subsequent stages, each defined as a named block in the `.mdx` file. The following summarises the remaining default stages in the Danish–English pipeline, though users are free to modify or replace them as required.

`group_examples` In this stage, the extracted examples are semantically grouped. The language model is instructed to cluster the examples by sense, assign each group a short Danish definition,[4] and place one to three representative examples under each category. The output introduces the structural elements `<gramcat>` (with a part-of-speech attribute), `<cat>` (with a sense number) and `<def>`.

[4] The definition is generated by the LLM solely on the basis of the grouped examples. If the pipeline were intended for monolingual dictionary production rather than bilingual output, it could easily include an additional stage focusing on refining or verifying sense definitions.

add_tran Here, the model adds idiomatic translations for each example. For every `<ex>` (example) line, it appends a corresponding `<trex>` (translated example) line in British English. The prompt explicitly requests natural, native-like phrasing, even at the cost of literalness. These translations remain aligned with the original examples and are grouped under the same `<trgrp>`.

extract_tran_lemma This stage identifies the English equivalent of the headword in each translated example. The model is asked to extract the relevant translation (e.g. *cow* for *ko*) and insert it as a new `<tran>` element inside each `<trgrp>`, placed before the original and translated examples. The dictionary form (e.g. the infinitive for verbs) is required, ensuring consistency with downstream dictionary structures.

reorder_dict Finally, the system restructures the entry so that all unique headword translations (`<tran>`) appear not only within their respective `<trgrp>` elements but also immediately after each sense definition (`<def>`). This duplication supports lexicographic tools that expect grouped translations per sense block, while preserving local context within the example groups.

Each of these stages produces valid XML according to its stage-specific DTD, and each step can be independently rerun, corrected or skipped. The modular design allows new stages – such as collocation extraction, etymology insertion or usage label tagging – to be added by defining additional prompt blocks in the `.mdx` file.

The `.mdx` prompt files may therefore evolve organically over time, adapting to new dictionary features or changing editorial priorities.

3.4.1 Manual Correction and Resuming the System

As an example, we have generated a dictionary entry for the Danish adjective *nørdet*.[5] In doing so, it was observed that several examples involving the verb *nørde* had been incorrectly included. If such issues occur systematically, the relevant prompt block in the `.mdx` file should be reviewed and refined. However, if the problem appears to be a one-off, an alternative is to manually edit the intermediate file – for example, `nørdet01_01.xml`, the output from stage one – and remove any erroneous examples, such as the following:

```
<trgrp>
<orig>I stedet for at gå til golf eller sejlads
nørder jeg igennem med bryggeri.</orig>
<ex>I stedet for golf nørder jeg igennem med bryggeri.</ex>
</trgrp>
```

After such corrections, the system can be resumed from this point using:

```
python lexicographer.py danish-english.mdx nørdet adjective \
   output/nørdet01_01.xml
```

[5] The past participle of the verb *nørde*, which is derived from the noun *nørd*, a borrowing from English *nerd*.

This approach allows for targeted adjustments without re-running the entire pipeline. Once the edited file is in place, the script continues with the next stage, incorporating the correction while retaining previous results. This facility allows for targeted adjustments without re-running the entire pipeline, enabling a semi-automatic workflow in which human editors can resolve specific anomalies without compromising automation.

3.5 Final Output

The final stage of the pipeline produces two parallel outputs: a structured XML file suitable for further processing or integration into dictionary databases, and a human-readable Markdown version designed for inspection and proofreading.[6]

The XML file contains a complete, machine-parseable entry for the given headword, including all example groups, definitions, translations, collocations and other dictionary-relevant components generated across the pipeline stages. Because each stage enforces XML validation against a Document Type Definition (DTD), the output is guaranteed to conform to a predictable and well-defined structure.

All files are written to the `output` directory. The naming scheme reflects the headword and run number, followed by either a stage number or the word `final`, depending on the file. For example, processing the adjective *nørdet* on run number 01 produces:

```
output/nørdet01_01.xml
output/nørdet01_02.xml
output/nørdet01_03.xml
output/nørdet01_04.xml
output/nørdet01_05.xml
output/nørdet01_final.xml
output/nørdet01_final.md
```

The run number allows the user to generate the same entry multiple times without overwriting previous results. The stage number indicates which step in the pipeline the file represents.

To view the Markdown file, one may, for instance, use Pandoc to produce a Word document:

```
pandoc -o output/nørdet.docx output/nørdet_final.md
```

These files may then be imported into a larger dictionary compilation workflow, used as input to further automation pipelines or reviewed by human editors. Since each intermediate stage is preserved and traceable, the resulting entry is not only reproducible but also open to revision and audit.

Additional output formats or editorial front-ends could easily be supported by modifying the final XSLT step.

[6] The Markdown version is created by a simple XSLT script; this can easily be modified to change the formatting or to produce other types of output, such as HTML.

3.6 Choice of Architecture

The pipeline was designed around XML and DTDs because this is the native format of IDM's DPS system used by the Danish Language Council, and integrating the LLM pipeline into this environment was the most practical solution. Nothing in the architecture, however, precludes adaptation to other infrastructures: the same modular stages could, in principle, just as easily write to a relational or graph-based database instead of XML files. We would welcome patches to the source code that make the system more general and flexible.

4. Evaluation

To illustrate the system's output, we present a sample entry generated for the Danish adjective *nørdet*.

nørdet *adjective*

6. (*meget interesseret i specifikke emner eller detaljer*) geek, nerd[7]
    - **Er man nørdet udi romantikkens klassikere, kan man høre et lille nik.** If you're a bit of a romantic literature geek like me, you'll catch a nod.
    - **Jeg har ellers været ret nørdet med den slags.** I've tended to be quite the nerd about that sort of thing.
    - **Hun er både ambitiøs og nørdet.** She's both ambitious and a bit of a nerd.
7. (*teknisk eller specialiseret på en særlig fokuseret måde*) geeky, nerdy
    - **Flere og flere får noget så nørdet som en sous vide.** More and more people are getting something as geeky as a sous vide.
    - **Det er egentligt ret nørdet, men også ret fedt.** It's actually quite nerdy, but also pretty cool.
    - **Det er lidt nørdet og følelsesmæssigt, forklarer hun.** It's a bit geeky and emotional, she explains.
8. (*person kan opfattes som genert eller socialt akavet*) nerdy
    - **Troels Lund Poulsen kan virke lidt nørdet.** Troels Lund Poulsen can come off as a bit nerdy.
    - **Mit yndlingscitat handler om en nørdet hukommelsescelle.** My favourite quote is about a nerdy memory cell.

For comparison, here is the result of a second run on the same input:

nørdet *adjective*

9. (*overdrevet interesseret i specifikke emner*) buff, geeky, nerdy
    - **Er man nørdet udi romantikkens klassikere, kan man høre et lille nik.** If you're a bit of a romantic literature buff, you might spot a nod.

[7] The system has chosen to translate the adjective using nouns here, matching the idiomatic phrasing of the examples. The prompts could be adjusted to prevent this, or an editorial note could be added to explain the rationale.

- **Jeg har ellers været ret nørdet med den slags.** I've been quite geeky about this sort of thing.
- **Mit yndlingscitat handler om en nørdet hukommelsescelle.** My favourite quote is about a nerdy memory cell.

10. (*lidt mærkelig eller socialt akavet*) socially awkward, oddball
    - **Troels Lund Poulsen kan virke lidt nørdet.** Troels Lund Poulsen can come across as a bit socially awkward.
    - **Hun er både ambitiøs og nørdet.** She's both ambitious and a bit of an oddball.
11. (*kompliceret eller teknisk i natur*) technical, techy
    - **Det er egentligt ret nørdet, men også ret fedt.** It's actually quite technical, but also pretty cool.
    - **Flere og flere får noget så nørdet som en sous vide.** More and more people are getting something as techy as a sous vide.
    - **Det er lidt nørdet og følelsesmæssigt, forklarer hun.** It's a bit technical and emotional, she explains.

The sample demonstrates that the system meets its intended goals:

12. **Consistency**: Entries follow a standardised format, making them easy to compare and integrate into a larger lexicographic project.
13. **Transparency**: All processing steps are preserved, with examples traceable to corpus data and outputs open to inspection and correction.
14. **Hallucination**: Artefacts are rare and typically arise from corpus or translation noise; genuine hallucination is minimal and easily caught.
15. **Originality**: Content is generated afresh, without replicating existing dictionary entries.
16. **Editability**: The output format is designed for direct use in a dictionary editing environment.

The translations are generally idiomatic, though often freer than a traditional dictionary might allow. More literal renderings can be obtained by adjusting the translation prompt.[8] Moreover, since concordance lines often lack full context, the model may misinterpret subtle discourse cues or thematic roles during shortening.

In sum, the system enables rapid prototyping of dictionary entries with clear separation of concerns, rigorous validation and human-in-the-loop flexibility. It offers a practical platform for experimentation, research and, potentially, production-level lexicography.

This remains ongoing work, and a formal evaluation with practising lexicographers has not yet been conducted. The pipeline and source code have been released publicly to invite testing and feedback from the wider community, which, together with ongoing testing at the Danish Language Council, will provide valuable insights for further refinement.

A typical dictionary entry using the `.mdx` file described here uses roughly 8,000–10,000 tokens per full run (five stages), corresponding at current prices to a cost of approximately 0.05–0.06 USD per entry.

[8] Prompt design remains an art as much as a science; getting the right balance often takes experimentation.

4.1 Scope and Limitations

Inflectional paradigms and pronunciation were deliberately excluded from the current system. These components are better sourced from established resources such as the Central Word Register of Danish (the "COR": Widmann (2023)) or Wikidata, rather than inferred via generative models. Future work may explore integrating structured lexical databases to support these features more robustly.

5. Conclusion

This paper has presented a functioning system for automatic generation of draft dictionary entries in XML format using ChatGPT. By decomposing the lexicographic process into a sequence of narrowly constrained, auditable steps, the system mitigates the well-known limitations of large language models in the areas of consistency, transparency, hallucination, originality and editability.

The system is fast, flexible and domain-agnostic. It supports dictionary creation for under-resourced languages, specialised domains and small corpora without sacrificing editorial oversight. Its architecture is fully language-independent, making it especially well suited to contexts where rapid prototyping is more valuable than exhaustive coverage.

While the system is designed to run fully automatically, it allows human intervention at any stage. This dual capacity enables both hands-off operation and fine-tuned human-in-the-loop editing – making the system adaptable to different workflows and editorial preferences. Crucially, it does not aim to replace human expertise but to elevate it: editorial judgement remains central, but the focus shifts from mechanical drafting to strategic decision-making and final validation.

In short: LLMs can assist, not replace. Structured workflows, corpus grounding and formal validation provide a pragmatic and sustainable path forward for the next generation of lexicography. By enforcing strict validation and traceability, the system also offers a model for responsible LLM use in dictionary production, reducing both ethical and legal risks.

To support reproducibility and reuse, the entire system – including Python source code, modular prompt files, DTD definitions and output formatting tools – is available on GitHub: https://github.com/twidmann/ai-lexicography. Released under the MIT Licence, the system is intended as both a demonstration of responsible LLM use in lexicography and a foundation for adaptation to other languages, dictionary types and editorial workflows.

5.1 Future Work and Outlook

Looking ahead, several avenues remain open for extension. The current system deliberately omits morphological data, pronunciation and etymology – areas better served by formal lexical resources such as the COR or Wikidata. Future versions may integrate these automatically where high-quality data exist, or flag them for manual post-editing.

Longer term, more speculative challenges also emerge. As AI systems increasingly assist or even replace humans in drafting, translating and interpreting texts, the audience and function of dictionaries may shift. Lexicographic output may need to be optimised not

just for printed dictionaries, but also for spoken ones (Tavast et al., 2023), interactive formats or those designed for LLM consumption. This raises foundational questions about the status of dictionaries in an AI-mediated linguistic landscape.

Paradoxically, we now possess the capacity to produce more dictionaries than ever before – rapidly, inexpensively and across countless domains – just as the traditional demand for standalone dictionaries may be waning. At the very least, the system presented in this article helps to lower the cost of producing them. Whether as a research tool, production engine or editorial assistant, it offers a flexible foundation for future lexicographic work.

Software

Attardi, G. (2015). WikiExtractor. https://github.com/attardi/wikiextractor.

de Schryver, G.M. & Joffe, D. (2023). The end of lexicography, welcome to the machine. https://www.youtube.com/watch?v=mEorw0yefAs. Paper presented at the 20th CODH Seminar, National Institute of Informatics, Tokyo, Japan.

Nichols, W. (2023). Invisible lexicographers, AI, & the future of the dictionary. https://www.youtube.com/watch?v=xYpwftj_QQI. Keynote presented at the eLex conference.

Rundell, M. (2023). Automating the Creation of Dictionaries: Are We Nearly There? In *Proceedings of the 16th International Conference of the Asian Association for Lexicography: Lexicography, Artificial Intelligence, and Dictionary Users*. Seoul: Yonsei University, pp. 1–9.

Tavast, A., Rundell, M., Rychlý, P., Kokol, M. & de Schryver, G.M. (2023). eLex ChatGPT Round Table. https://www.youtube.com/watch?v=dNkksTDYa_s. YouTube video.

Widmann, T. (2023). The Central Word Register of the Danish language. In *Electronic lexicography in the 21st century (eLex 2023): Invisible Lexicography. Proceedings of the eLex 2023 conference*. Brno: Lexical Computing CZ s.r.o., pp. 91–103. URL https://elex.link/elex2023/wp-content/uploads/elex2023_proceedings.pdf#page=91.

Widmann, T. (2025a). ai-lexicography: Corpus-based lexicography using the OpenAI API. https://github.com/twidmann/ai-lexicography.

Widmann, T. (2025b). ChatGPT som leksikograf: Ordbogens fremtid? In *20. Møde om Udforskningen af Dansk Sprog*. Aarhus, Denmark: Institut for Kommunikation og Kultur, Aarhus Universitet, pp. 427–442. Presented October 2024.