Emil.RuleZ! – An exploratory pilot study of handling a real-life longitudinal email archive

Balázs Indig[1,2,5], **Luca Horváth**[2,5], **Dorottya Henrietta Szemigán**[2,3], **Mihály Nagy**[2,4]
[1]Eötvös Loránd University, Department of Digital Humanities
[2]National Laboratory for Digital Humanities
[3]Eötvös Loránd University, Doctoral School of Literary Studies, Comparative Literature Doctoral Program
[4]Eötvös Loránd University, Atelier Department of Interdisciplinary History
Muzeum krt. 6-8., H-1088, Budapest, Hungary
[5]Eötvös Loránd University, Doctoral School of Informatics
Pázmány Péter stny. 1/C, H-1117, Budapest, Hungary
{indig.balazs,horvath.luca,szemigan.dorottya,nagy.mihaly}@btk.elte.hu

Abstract

An entire generation that predominantly used email for official communication throughout their lives is about to leave behind a significant amount of preservable digital heritage. Memory institutions in the USA (e.g. Internet Archive, Stanford University Library) recognised this endeavor of preservation early on, therefore, available solutions are focused on English language public archives, neglecting the problem of different languages with different encodings in a single archive and the heterogeneity of standards that have changed considerably since their first form in the 1970s. Since online services enable the convenient creation of email archives in MBOX format it is important to evaluate how existing tools handle non-homogeneous longitudinal archives containing diverse states of email standards, as opposed to often archived monolingual public mailing lists, and how such data can be made ready for research. We use distant reading methods on a real-life archive, the legacy of a deceased individual containing 11,245 emails from 2010 to 2023 in multiple languages and encodings, and demonstrate how existing available tools can be surpassed. Our goal is to enhance data homogeneity to make it accessible for researchers in a queryable database format. We utilise rule-based methods and GPT-3.5 to extract the cleanest form of our data.

1 Introduction

We live in a time when many people's email correspondence is preserved as a digital legacy. As these people have mostly used email for official communication throughout their lives (because of their habits) it is possible to look back over the majority of their electronic written communication (Jaillant, 2019). However, such digital email legacy raises a number of moral and legal questions. For example, data protection and privacy legislation, such as the General Data Protection Regulation (GDPR) (European Union, 2016), demands a challenging compliance process and encourages institutions not to implement long-term preservation for their own safety, which may become a relevant issue in the future. Furthermore, in most cases, there is not necessarily a complete separation between private and corporate emails (Cocciolo, 2016; Srinivasan and Baone, 2008). While from the technical perspective, email has undergone many changes since its inception and has become heterogeneous in terms of standards and implementation (Partridge, 2008), which makes longitudinal archives difficult to process, analyse and aggregate.

While different tools can be used to create e-mail archives in different formats (Digital Preservation Coalition and Prom, 2019) in our case it is assumed that the mail archive is already available as an MBOX file (e.g. Google Takeout or similar services). We got legal authorisation to use (without publishing partially or fully) a real-life email archive of a deceased public figure's correspondence (Hungarian, Romanian, English) for our pilot study to uncover and solve possible technical difficulties. Our aim was to create a methodology that could successfully process a real-life MBOX file that contains a longitudinal correspondence and produce an output that could be searched, visualized, and analyzed by researchers interested in the author's official communication.

2 Evaluation of the Available Tools

Several tools exist for processing MBOX archives and all of them are built by adopting alternative approaches with specific perspectives in mind. Since these approaches are very diverse, each tool has its own strengths and weaknesses which should not be ignored when pursuing our goal (Carlson, 2020). In this section, we introduce a selection of the tools we evaluated (divided into two classes) before we decided to write our own.

For each test we used two MBOX files. One is an artificially created demo MBOX file (Willison, 2022b) containing only two emails. It lacks misspelled emails, notifications, circulars, returned error messages from mailer-daemons (i.e. mail delivery systems), etc. The other is the aforementioned real-life MBOX file that we want to process covering more than one decade and therefore containing many of the errors mentioned above.

All examined tools more or less handle the important metadata headers (FROM, TO, CC, BCC, SUBJECT, DATE, etc.) but most of them fail when it comes to decoding the textual data that particular email bodies contain. We can classify these tools into two main classes: a) those often abandoned and only half done, and b) the complex monolith solutions of institutions that are hard to use without appropriate expertise. We will use this classification in the rest of this chapter.

The first group of programs (Vestal, 2018; Willison, 2022a; Sharma and Bhattacharya, 2023; Mineev, 2021; Juopperi, 2016) promises to process MBOX files and insert the output into various data formats (e.g. CSV, JSON, SQL) for further processing. For the artificial data set, this group of tools worked well but with our non-artificial test data, they failed to produce usable output (i.e. contained raw undecoded string fragments e.g. in *Quoted-Printable* or *Base64* form) if any. We examined the root cause of the errors and it turned out that these programs apply false assumptions and are beyond repair. In general, if one wants to process a very complex data set, their use is not recommended without technical expertise, and we decided it was better to start with a clean slate.

For people without technical skills, viewing an MBOX file in *Mozilla Thunderbird* or *MBOX Viewer* is a great opportunity to interact with the data (i.e. read it), as these tools are halfway between the two mentioned groups: they can handle non-artificial data and do not require expertise (neither technical nor archiving) to operate. However, we found that the export functionality of MBOX Viewer is half broken: it can produce a CSV file for our non-artificial data set, but we could not properly load it with *MS Excel* or *LibreOffice Calc* probably because the garbled delimiters, limiting further deeper analysis. We assumed that it did not escape characters with syntactical meaning in tabular format, breaking the data structure.

In the second group, we tested two well-established solutions that promise more than extracting MBOX to common data formats. *Mailbagit* (University at Albany, 2023b) can import various kinds of email archives and convert them to *MAILBAG* format (University at Albany, 2023a) while exporting the data to other formats (TXT, WARC, PDF, etc.). For our non-artificial data set it yielded a lot of error messages, but could produce a good CSV file with the mandatory metadata and individual files for the payload of each message. When necessary it uses automatic character encoding detection but detects absurd encodings (e.g. Turkish code page), and there is no way to correct such mistakes manually due to its complexity.

ePADD (Stanford University's Special Collections, 2021; Schneider et al., 2019) is developed by the Stanford University Library primarily for English email archives. It offers various features for importing data and allows you to choose how and from where to import it. The same errors were identified as in Mailbagit, but it offered no possibilities for manual repair. It is a comprehensive solution when it comes to email archiving and is the most advanced tool we could find for those who possess archiving expertise but lack technical skills.

In the long run, the bugs may be fixed in some of the aforementioned programs by their maintainers. However, as they did not fit our primary goals, we decided to implement our own lightweight solution for this specific archive which can be easily extended with little technical skills if an error occurs while processing other archives.

3 Our method

To accommodate storing a sequence of multiple emails, the format of raw MBOX data contains encoded fragments, i.e. one-byte long ASCII characters with no syntactical meaning chunked into 80-byte long lines, which allows easy handling. Non-ASCII byte sequences are encoded by a *binary-to-text encoding* scheme such as *Quoted-Printable* or *Base64* which when decoded returns the original values (i.e. string or binary) of the individual emails. One email record is composed of multiple case-insensitive key-value-style (standard and non-standard) headers and the payload consisting of recursively embedded payload parts (e.g. HTML, text, or binary attachment). To get the character representation of non-ASCII, byte-represented text segments, bytes need to be decoded with the supplied character encoding. In some cases, no or wrong character encoding was specified e.g. binary data erroneously has character encoding. We used the built-in *email* module of Python which contains utility functions for most of the aforementioned steps. Proceeding carefully from the headers to the payload we realised they posed different challenges, therefore in the following, we discuss them separately.

3.1 Headers

We gathered the frequency distribution of all headers and their values in a case-insensitive manner. This showed us the irregularities of the data and the non-standard headers which were to be normalised or ignored. We classified headers into three types which we identified by the name of their keys: a) date, b) plain text, and c) address list.

The date values in some cases contained localised human-readable statements on the timezone – sometimes with additional character encoding errors – or contained timezone information in a form that was not handled by Python (-0000 instead of +0000 for UTC time). Fortunately, there is a specific built-in function (*email.utils.parsedate_to_datetime()*) in the email module that handles all but the negative UTC case which we replaced beforehand.

The *email.header.decode_header()* built-in function did the heavy lifting on decoding the above-mentioned binary-to-text encoded plain text chunks while keeping the data in bytes form with their specified character encoding because only binary-to-text encoding is safely decipherable. The returned chunks had to be converted to character strings with our own code using the supplied character encoding and handling possible encoding errors. In some cases, the built-in function did not return any character encoding. This either meant strings were already decoded (i.e. were in string type) or it was left up to us to interpret the remaining bytes-type part (in our case all of which were in ASCII). Finally, string chunks needed to be concatenated to restore the original value.

The address sequences could be uniformly split with the *email.utils.getaddresses()* function to name-address pairs (the format defined in the email standards). However, this function leaves the decoding of the binary-to-text data to the user, therefore, the aforementioned decoding heuristics had to be reused here.

These methods cover the common header types, which are required for average use cases. Our program lets the user include or map non-standard headers at will for special use cases (e.g. thread-id, delivered-to, etc.) as they probably do not need extra decoding steps.

3.2 Payload

The email payload is recursively built from *parts*. Nowadays, most emails' body is in HTML, but have a plain-text variant as a separate part which may or may not represent the same textual content as the HTML. Binary parts are also common due to attachments or inline elements (e.g. images) which HTMLs are often augmented with. These components are stored as individual payload parts, but are difficult to distinguish them. As our goal was to extract text only we could ignore attachments and inline binary blobs. However, according to the standards only one of the HTML or plain text content is required – but both are allowed and commonly used side by side –, therefore we decided to keep all textual information and examine them later.

For each payload part, we created a table of values of the available features (which can be extracted by the built-in functions) to define the behaviour of our program by inspecting the groups of values. The used features are listed in Table 1.

Name	**Value**
filename	str/None
is_multipart	True/False
content_type	MIME-type/None
payload	bytes+encoding/None
content_charset	str/None
content_disposition	str/None
has_parts	True/False

Table 1: Features used to classify payload parts

We found significant connections between the features. Some were expected[1], but others were not. For example, *content_disposition* turned out to be unusable as it had inconsistent values, therefore,

[1] *is_multipart* and *has_parts* had the same values: when they were True *payload* was None.

we used the *filename* instead. Besides that, *content_type* were missing in some cases, so we had to utilise *libmagic* to detect one. We broke down the data set into the following classes (Table 2.):

Class name	**Action**
multipart	recurse on subparts
filename is not None	attachment, ignore
content_charset is None and not text MIME-type	inline image, ignore
content_charset is None and text MIME-type	text w/o encoding
content_charset is not None and strange MIME-type	textual data (CSV, iCal, etc.)
not multipart and no filename and content_charset is not None and text MIME-type	proper text

Table 2: Payload class-action pairs

4 Using GPT for Curation

The remaining problems turned out to be the result of non-standard behaviour, which was easier to solve with a solution more capable of handling semi-structured data, therefore we chose to use *GPT-3.5* (Ouyang et al., 2022) (to facilitate affordable reproducibility) adopting a few-shot methodology (see Appendix A.1.). Upon replying to, or forwarding an email the old text is separated by the main metadata of the email (in string form formatted in a non-standard, language-specific way) from the new text which is usually written at the top of the email body, however, in some cases at the discretion of the user, the reply is inlined (inserted between the lines of the old email). This process often results in concatenated email-body texts requiring separation. To solve these issues, we identified the following tasks: a) decoding the use of non-standard string form metadata inside the email body, and b) separating concatenated email-body texts. Furthermore, we also used GPT-3.5 to fix the remaining edge-case encoding errors that were caused by "lossy" decoding (e.g. replacement characters, omitting faulty byte sequences, incorrectly decoded characters).

GPT-3.5 performed best on metadata extraction, where it was able to extract FROM and TO addresses, dates, and SUBJECT strings even with simple prompts. Handling concatenated email-body texts, however, proved to be a more difficult task, therefore, for testing their separation accuracy, we chose 100 examples with varying numbers of previous email text recursively included in the payload. 77 were successfully separated, failing mostly on inputs with more complex text structures that did not contain proper helper annotation (e.g. ">" used for indenting previous email text lines) (see Figure 3. in Appendix A.1.).

We found that many of the erroneously encoded emails that were left had the same problem: an automatic mechanism (antivirus) had pasted a footnote to the payload, but with wrong encoding, which caused the decoding of the whole message to fail. Although GPT-3.5 successfully handled these cases, we also implemented a rule-based method of splitting the text and applying another encoding to the footnote part, fully eliminating this type of encoding problem. Only a few complex encoding errors remained that neither GPT-3.5 nor a rule-based method could solve, as they were products of several layers of incorrect processing, and the resulting character combinations were indecipherable even for humans. Our experiments show that a rule-based workflow could potentially be expanded by using Large Language Models, if tasks are well compartmentalised and split into separate problem areas.

5 Visualising the Resulting Data

With the data cleaned and normalised to the limit, as a final step to facilitate access to authorised digital humanists who prefer visual representations of data, we loaded it into an off-the-shelf application suitable for n-gram based data exploration (*N-gram Trend Viewer* (Indig et al., 2022)). One example of exploring the metadata-rich text-based corpora – using only metadata that is safe from compromising GDPR – is the frequency of different email providers that the owner of the account interacted with over time (see *Figure 1*.). Naturally, those who have legal access to the data can make more in-depth queries that the system supports.

6 Evaluation and Conclusion

Our experiments with the automatic character encoding recognizer systematically resulted in Turkish code pages, which can be safely ruled out from the set of possibly used languages and code pages, therefore, we opted to manually observe each oc-

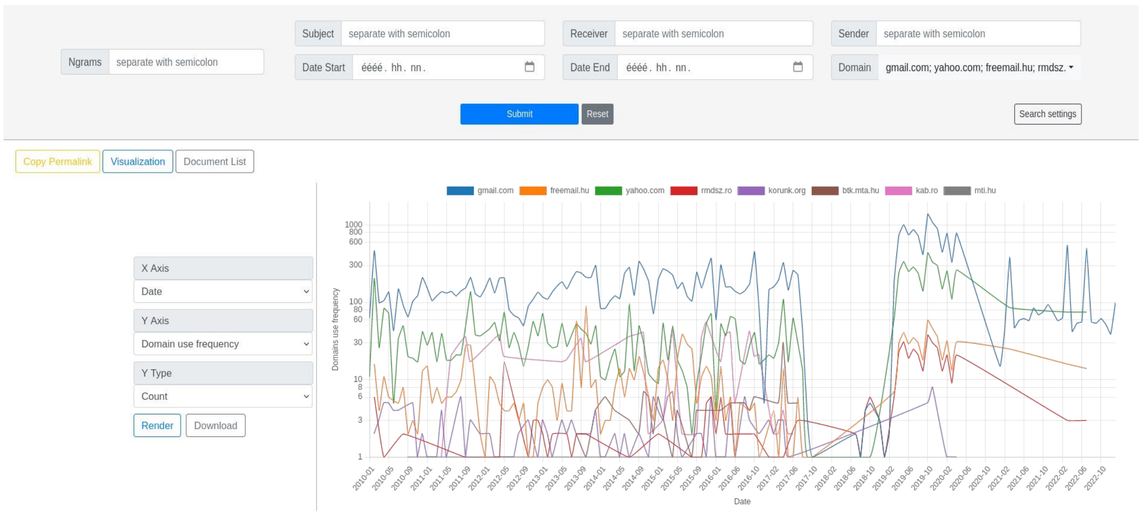
Ngrams: separate with semicolon
Subject: separate with semicolon
Receiver: separate with semicolon
Sender: separate with semicolon
Date Start: éééé. hh. nn.
Date End: éééé. hh. nn.
Domain: gmail.com; yahoo.com; freemail.hu; rmdsz.
X Axis: Date
Y Axis: Domain use frequency
Y Type: Count

Figure 1: Usage count of selected anonymized email domains (FROM, TO, CC, and BCC) over time. There was a drop in traffic around 2017 when the subject passed away, however, strangely enough after a period of inactivity the account started interacting with several domains again, most likely due to someone gaining access.

currence and guess the most likely encoding. With the described payload classification only a few (37 from 11,245) email payload parts remained that had trivially erroneous encoding specified, or failed to decode with the specified/suggested encoding. This could be further reduced by using GPT and the string-splitting technique. The existing methods we evaluated found the same number of character encoding errors as our method did, however, the final error rate was worse for the evaluated methods due to the automatic mechanisms and the lack of possibility for correction.

Our method and ePADD both found that many email addresses had different names associated with them, which enables these names to be added as an alias along with a canonical name to a semantic database (e.g. the "Also known as" field in Wikidata (Vrandečić and Krötzsch, 2014)) for later use. ePADD used English word lists with little success to recognise named entities in the text. The lists can be changed but due to the monolithic nature of the program, the clearly not state-of-the-art method cannot.

We conclude our pilot project a success[2], as we recovered most of the errors and created an intuitive WebUI for the MBOX data to help researchers explore the email archive. To open up more possibilities in the future, the conversion of emails to standard TEI XMLs (DeRose, 1999) is an option worth exploring as it could additionally handle the complex philological aspects of inline replies. Trying other MBOX files is also desirable to make our tool more robust and handle more non-standard headers since it was built with extensibility and customisability in mind.

[2]The code is published under GPL 3.0 license at `https://github.com/elte-dh/mbox-parser`.

Acknowledgements

We extend our special thanks to András Hajós for coining the name of his band, paying homage to the beauty of the email. We hope to express our appreciation by choosing our suggestive title.

This work is supported by the *DKOP-23 Doctoral Excellence Program* of the *Ministry of Culture and Innovation, Hungary* from the source of the *National Research, Development and Innovation Fund*.

MINISTRY OF CULTURE
AND INNOVATION

References

M. E. Grenander Department of Special Collections & Archives University at Albany. 2023a. Mailbag specification (1.0). `https://github.com/UAlbanyArchives/mailbag-specification`.

M. E. Grenander Department of Special Collections & Archives University at Albany. 2023b. Mailbagit.

https://github.com/UAlbanyArchives/mailbagit. Last accessed on 25/09/2023.

Clare Carlson. 2020. One size does not fit all: Exploring email archiving workflows. School of Information and Library Science (master's thesis).

Anthony Cocciolo. 2016. Email as cultural heritage resource: appraisal solutions from an art museum context. *Records Management Journal*, 26(1):68–82.

Steven DeRose. 1999. XML and the TEI. *Computers and the Humanities*.

Digital Preservation Coalition and Christopher J. Prom. 2019. Preserving email 2nd edition. *DPC Technology Watch Report*, 28.

European Union. 2016. Council regulation (EU) no 679/2016. https://eur-lex.europa.eu/legal-content/EN/TXT/PDF/?uri=CELEX:32016R0679.

Balázs Indig, Zsófia Sárközi-Lindner, and Mihály Nagy. 2022. Use the metadata, Luke! – an experimental joint metadata search and n-gram trend viewer for personal web archives. In *Proceedings of the 2nd International Workshop on Natural Language Processing for Digital Humanities*, pages 47–52, Taipei, Taiwan. Association for Computational Linguistics.

Lise Jaillant. 2019. After the digital revolution: working with emails and born-digital records in literary and publishers' archives. *Archives and Manuscripts*, 47(3):285–304.

Jari Juopperi. 2016. E-mail message to JSON converter. https://github.com/jmjj/messages2json. Last accessed on 13/09/2023.

Vsevolod Sebastian Mineev. 2021. Python script to extract emails from an .mbox file. https://github.com/vsevolod-mineev/csv-from-mbox. Last accessed on 13/09/2023.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Advances in Neural Information Processing Systems*, 35:27730–27744.

Craig Partridge. 2008. The technical development of internet email. *IEEE Annals of the History of Computing*, 30(2):3–29.

Josh Schneider, Chance Adams, Sally DeBauche, Reid Echols, Callum McKean, Jessica Moran, J., and Dorothy Waugh. 2019. Appraising, processing, and providing access to email in contemporary literary archives. *Archives and Manuscripts*, 47(3):305–326.

Prakhar Sharma and Adrita Bhattacharya. 2023. MBOX to JSON. https://github.com/PS1607/mbox-to-json. Last accessed on 13/09/2023.

Arvind Srinivasan and Gaurav Baone. 2008. Classification challenges in email archiving. In *Rough Sets and Current Trends in Computing*, pages 508–519, Berlin, Heidelberg. Springer Berlin Heidelberg.

University Archives Stanford University's Special Collections. 2021. ePADD, email Process Appraise Discover Deliver. Last accessed on 25/09/2023.

Allan James Vestal. 2018. mbox-tools. https://github.com/DallasMorningNews/mbox-tools. Last accessed on 13/09/2023.

Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A free collaborative knowledgebase. *Commun. ACM*, 57(10):78–85.

Simon Willison. 2022a. mbox-to-sqlite. https://github.com/simonw/mbox-to-sqlite. Last accessed on 13/09/2023.

Simon Willison. 2022b. Sample MBOX. https://github.com/simonw/mbox-to-sqlite/blob/main/tests/enron-sample.mbox. Last accessed on 13/09/2023.

A Appendices

A.1 The Used Prompt for Email Separation

MODEL
gpt-turbo-0613
TEMPERATURE
0
SYSTEM
You are a [LANGUAGE] email analysis assistant.
Your job is to take an email body text and if it contains the text of previously sent emails as an email correspondence,
then separate it into individual emails, and finally return the separate emails annotated with ”SEPARATE_EMAIL:”. You also need to remove any characters
that are in some cases added to the email body text to annotate forwarded emails,
emails sent as replies, or references to the original email.
These characters are usually a greater-than character: ”>”.
Not all previous correspondences are annotated with a greater-then character.
If encounter strings following the format ”[DATE] [NAME] wrote, [EMAIL]:”
then leave it in the output as if it were part of the email text.
Some input texts may be a single email, others may be a sequence of emails
that contain the latest email and other previously sent emails that are replies, forwards,
or the original message.
USER-ASSISTANT pairs of example email and example email with separation tags (x 3)
USER {input_email_text}

Table 3: Example prompt details of *OpenAI chat API* requests for separating augmented email payload texts, with a few-shot approach. USER-ASSISTANT email pairs are omitted for privacy reasons. Fourth *USER input_email_text* is to be replaced for each request with the actual email payload text.