CAT: An LLM-based Tool for Content Analysis in Experimental Economics

Working Paper #0119

Andrzej Baranski, David J. Cooper and Jeong Kyu Lee

NYU Abu Dhabi

August 2026

جامعة نيويورك أبوظبي

NYU ABU DHABI

CAT: An LLM-based Tool for Content Analysis in Experimental Economics*

Andrzej Baranski † David J. Cooper ‡ Jeong Kyu Lee §

August 14, 2026

We introduce the Communication Annotation Tool (CAT), an LLM-based tool for content analysis. CAT is intended to analyze free-form communication from economic experiments, accommodating a wide variety of communication structures. We describe CAT's features including a broad array of customization options and a template for developing a coding manual. A detailed guide for using CAT is provided, including a step-by-step demonstration showing how CAT is used to quantify the content of communication from an experimental dataset. CAT is designed for easy monitoring of the coding process which ensures replicability of the communication coding method and simplifies identification and resolution of problems. We conclude by discussing CAT's limitations and issues that users should keep in mind.

Keywords: LLM, content analysis, communication, coding, research software, annotation tool

*Andrzej Baranski gratefully acknowledges financial support from Tamkeen under the NYU Abu Dhabi Research Institute Award CG005. The authors are grateful for the support from the Social Science Experimental Laboratory at NYUAD

†Division of Social Science & Center for Behavioral Institutional Design, NYU Abu Dhabi, email: *a.baranski@nyu.edu*.

‡University of Iowa and University of East Anglia, email: *david-j-cooper@uiowa.edu*.

§NYU Abu Dhabi Social Science Experimental Laboratory, email: *jkl499@nyu.edu*.

1 Introduction

Free-form written communication is a valuable source of process data for experimental economists. Content analysis of messages has helped researchers gain insights about subjects' preferences, their reasoning processes, and the mechanisms by which communication affects behavior and outcomes (Brandts et al., 2019). In spite of its value, use of content analysis has been limited by the cost, in terms of both money and time, of quantifying communication data ("coding"). A typical coding project takes months to complete and involves paying for many hours of research assistance. LLM-based coding promises to stimulate use of content analysis by decreasing the costs.

Unfortunately, effective use of LLM-based coding requires a level of technical expertise that many researchers lack. Even for technically adept researchers, the fixed costs of *carefully* implementing LLM-based coding are high. Ideally, a researcher should develop a coding manual, pre-process the raw communication data, convert the coding manual and experimental context into prompts, program a script, connect it to one or more APIs, manage the API credentials, convert the model responses into a usable dataset, handle errors, implement repeated runs, test and debug the process before implementation, validate the results, and document the entire process for replicability. LLM-based coding is an arduous task. This raises a concern that increased use of LLM-based coding will reduce methodological rigor, possibly due to a lack of knowledge about the relevant methodology or a desire to make the coding process easier. It would be unfortunate if LLM-based coding made content analysis of communication data more accessible but reduced the quality of this analysis.

The purpose of this note is to introduce the Communication Annotation Tool (CAT), an LLM-based tool for content analysis that implements the methodology for LLM-based coding introduced by Baranski et al. (2026). CAT largely automates the use of LLM-based coding, making it accessible to inexperienced LLM users while reducing implementation costs for experienced users. It also automatically produces a replication package, further reducing researchers' workloads and increasing transparency of the coding process. CAT provides researchers with a straightforward way to implement LLM-based coding, adhering to established methodological standards.

To understand what CAT does, it helps to first describe how RA-based coding is typically done. The process starts with researchers developing categories that represent major themes in the communication data (e.g., all messages in which subjects make promises about their

future behavior). The categories are explained to a team of research assistants ("RAs") who then independently assign blocks of text to categories.

Coding by RAs is the part of the coding process that generates most of the costs. CAT replaces this part of the coding process; rather than having RAs code the communication, the LLM uses materials provided by the researchers to do the coding.[1] It is designed to be easy to use for novices while providing power and flexibility to experienced users. Researchers set up a coding task through an intuitive graphical interface. No software installation is required; users only need to have a browser and provider credentials for their chosen commercially available LLM(s). Detailed knowledge of Python, LLMs, or APIs is not required to use CAT. The graphical interface largely automates tasks like formatting the dataset. The details of coding exercises vary widely depending on the structure of the dataset and the goals of the researchers. CAT is highly customizable to handle a wide variety of coding tasks. For example, the researchers have broad flexibility in choosing the unit of coding (individual messages, bilateral conversations, etc.); CAT automatically creates the necessary variables that allow LLMs to identify different communication episodes. Likewise, the researchers control what LLMs are used, the number of runs per LLM, and the technical details of how the models are implemented (e.g., setting the temperature parameter). CAT integrates the results of the coding exercise with the original dataset and outputs an easy-to-use CSV file. It also has the ability to create a replication package that can be submitted to a journal.

Our primary goal in creating CAT is to help experimental economists conduct methodologically sound LLM-based content analysis. This paper necessarily contains large amounts of technical material, but we also include a step-by-step demonstration showing how to use CAT to code a dataset. The website also features a wide variety of training materials — new users may find the training video giving a step-by-step demonstration of CAT especially useful. The material on the website will be updated as CAT evolves, allowing users to easily keep up with the latest developments.

CAT is available at `https://ssel.chatkit.nyuad.nyu.edu/`. We request that any researchers who make use of CAT cite Baranski et al. (2026), the paper that introduced the methods implemented by CAT.

The remainder of this paper is organized as follows. Section 2 describes how CAT is used to code communication data from an economic experiment. Section 3 provides a step-by-step demonstration designed to train researchers on how to use CAT. Section 4 covers the

[1] For an LLM-based method of developing categories, see Cooper et al. (2026).

technical aspects of data handling and procedures. Section 5 discusses our tool limitations and concludes.

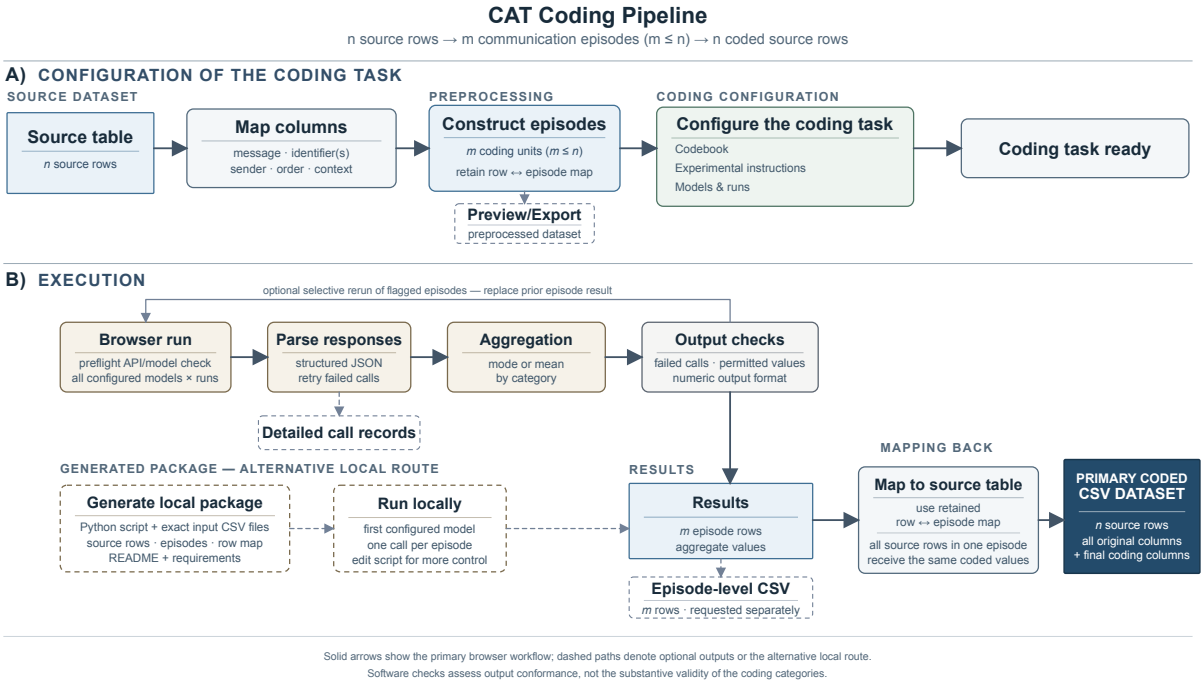

Figure 1: CAT pipeline diagram

2 Using CAT

2.1 Summary

This subsection summarizes the process of coding a communication dataset using CAT. Figure 1 illustrates the full coding pipeline and the subsequent subsections provide a more detailed description for each step of the process. The upper portion of the figure shows how the source dataset and coding configuration are used to construct the coding task, while the lower portion shows execution, aggregation, and output checks.

The set-up for coding a dataset using CAT is a four-step process.

1. After uploading their dataset, the researchers identify the variables that determine the unit of coding, henceforth referred to as a "communication episode." For example, a communication episode in a multi-lateral bargaining game might be identified by

which subject sent the messages and what round they were sent in. CAT automatically generates a variable that identifies each unique communication episode.

2. The researchers use CAT to create coding categories. CAT does not develop categories for researchers, but includes a template researchers can use to either import existing categories (we recommend cutting and pasting) or create new categories. The template allows researchers to include detailed instructions about how the category should be coded, examples that illustrate correct coding, and additional context to help the LLMs interpret the category. CAT lets researchers choose a variety of formats for how a category is coded (binary, categorical, numerical scales, and free-form text). If the researchers choose to use multiple LLMs or multiple replications per LLM, the definition of the category also specifies how these multiple runs are aggregated into a single coding.

3. The researchers upload their experimental instructions, providing the LLMs with additional context for coding.

4. The researchers configure the LLMs used for the coding. This includes choosing which LLMs they wish to use,[2] providing an API key (CAT is great, but you still need to pay for your tokens!), deciding how many replications of the coding should be performed for each LLM, and choosing parameters for the models (e.g., the temperature parameter).

Before the LLMs code the communication data, CAT automatically validates the model credentials and configurations. It then creates a Python script which is submitted to the APIs for the chosen LLMs. While the script is running, users can view communication episode-level progress and results. After the coding is completed, any communication episodes that require additional attention (errors, abnormal responses, API failures, etc.) are flagged. CAT can recode selected communication episodes rather than repeating the entire dataset, generating substantial savings if only a small number of communication episodes have issues that require recoding.

CAT outputs a CSV file that merges the coding output into the original dataset, eliminating the need to merge datasets. Researchers have the option of compacting the dataset to only include one row per communication episode. CAT can also generate a replication package for submission to a journal. This package includes a Python script, three CSV files

[2]CAT currently supports multiple models from OpenAI (ChatGPT), Google (Gemini), DeepSeek, Anthropic (Claude), and xAI (Grok). We plan to expand to additional providers in the near future.

(parsed source rows, constructed communication episodes, and row-to-episode mapping), a README file, and a requirements txt file. Researchers can also use these materials to reproduce (and modify if so desired) the analysis done by CAT outside the graphical interface.

2.2 Uploading and Structuring the Dataset

The researchers upload the communication dataset as a CSV or Excel file. Each row typically contains a message and may also contain additional columns such as identifiers, sender information, message order, or any additional context.

After the data is uploaded, CAT opens the *Map Columns* window. In CAT, *Map Columns* means selecting columns from the uploaded dataset and assigning them to the roles used by CAT: *Message*, *Episode Identifier*, *Sender*, *Order*, and *Context*. Mapping does not change the source dataset; it tells CAT how each column should be used. This window is used to define how the rows of the dataset will be grouped into communication episodes. A *communication episode* is a set of messages that are coded as a single coding unit. Communication episodes vary widely between coding exercises. A communication episode might consist of single messages, all messages exchanged through the same channel, or a collection of messages sent by one sender.

The *Map Columns* window first asks the researchers to identify the column in the dataset (*Message*) that contains the text to be coded. The researchers then select the variables that define a communication episode (*Episode Identifier*). For example, suppose a communication episode consists of all messages sent by a single subject in a round of the game being played. The researchers would click on the variables identifying the subject and round, and then CAT automatically generates a new variable that identifies unique combinations of these two variables. Researchers have the option to select "*Each row is its own episode*" when no grouping of rows into communication episodes is needed.

As an option, researchers can select a variable (*Sender*) that identifies who produced each message. For multi-participant conversations this is required when a category will be coded separately for different participants. Another optional variable (*Order*) identifies a timestamp or turn-number column and determines the sequence in which messages are presented within a communication episode. This is useful for the LLMs in interpreting communication episodes that consist of a conversation with multiple messages. When multiple rows have the same value for *Order*, CAT preserves their relative order in the uploaded dataset. Researchers can use *Context* to identify additional columns that the model should consider,

such as the treatment condition or actions taken by subjects. The researchers can provide a description explaining the meaning of each context field. Because a communication episode contains a single communication episode-level value for each contextual field, every selected *Context* column must contain exactly the same value in every source row belonging to a communication episode.

When the researchers click on *Save & Proceed*, CAT checks every selected *Context* column within the proposed communication episodes. If a communication episode contains different values, including a mixture of blank and nonblank values, CAT displays a warning that identifies the affected variable, reports the number of conflicting communication episodes, and provides an example of the conflicts. The researchers will then need to correct and re-upload the dataset or unselect the inconsistent *Context* variable. After passing this check, CAT groups the source rows according to the variables selected as *Episode Identifier* and (if relevant) *Sender* and displays a preview of the resulting communication episode-level dataset. The preview allows the researchers to verify the number and composition of the communication episodes before execution. The preprocessed communication episode table can also be downloaded as a CSV file.

2.3 Creating the Coding Manual

To create a coding manual, the researchers first must specify how CAT should handle communication episodes with no message text. Selecting *Ignore* prevents a coding request from being made for an empty communication episode. The corresponding source rows remain in the output, but their coding fields are left blank (i.e. missing observations). Selecting *Code as Value* means that CAT asks the model to apply the category definitions to the communication episode. Generally, this will lead to no categories being coded.[3]

The researchers then construct the codebook, defining the categories to be coded. CAT provides a template that the researchers fill out for each category. They are asked to give a name and a definition for the category. For researchers with an existing codebook, this information can simply be pasted into the appropriate slots. The researchers must choose one of four output types for each category: A *Binary* category uses the fixed values 0 and 1. A *Categorical* category uses a researcher-defined set of permitted values (e.g., Low, Medium, and High). A *Numeric* category returns a number (e.g., a number in the range from 0 to

[3]For example, suppose a category is created that codes whether a message contains a promise. In the absence of any text, no promises can be sent and the category will be coded as a zero.

10), and a *Text* category returns a free-form textual response. For binary and categorical codings, the researchers need to provide definitions for each possible value. Typically these sub-definitions will expand on the primary definition for the category. There is also the option to provide examples for each possible value (e.g., an example of text that contains a promise) and to provide additional context explaining how the value should be applied in particular circumstances. For example, the researchers could specify that a statement counts as a promise only when the sender commits to a future action, whereas a request, prediction, or expression of hope does not. Additional context can also explain study-specific terminology or shorthand. For example, the researchers might specify that '15–15' denotes an allocation of 15 units to each participant in a multilateral bargaining game, or that, when the total amount being split in a bargaining game is 20, a proposal such as 'you take 10 and I take the rest' represents an equal split. In general, LLMs do a better job of coding if detailed definitions, multiple examples, and clear context are provided. *Numeric* and *Text* categories do not have a fixed value list, so their category definitions must explain the expected output.

The researchers can choose whether each category is coded once per communication episode or separately for each sender within each communication episode. In the latter case, CAT displays the sender names detected automatically from the *Sender* variable (see Section 2.2). The researchers must verify this read-only list before saving the codebook. Sender-level coding cannot proceed if no *Sender* variable has been selected, if that variable contains blank sender values, if no sender names are detected, or if the detected list has not been verified. Changing the dataset or *Sender* variable resets this verification.

For each category, the researchers also select how outputs from repeated model calls will be aggregated: *Majority Vote* (mode) or *Average* (mean). CAT defaults to *Majority Vote* for *Binary* and *Categorical* categories and *Average* for *Numeric* categories.

These definitions and settings form the coding manual supplied to the LLMs and the rules used to summarize repeated outputs. The completed coding manual can also be exported in several common formats for review, reuse, or inclusion in project documentation and replication packages.

2.4 Experimental Instructions

The researchers upload all instructions that experimental subjects received, including the tasks, roles, payoffs, and communication rules relevant to the study. CAT supplies these instructions to the LLM as background information; they are not themselves treated as

messages to be coded, but they aid in interpreting the coding categories.

The instructions can either be pasted directly into the text box under the Experimental Instructions tab or imported as a PDF file. For PDF import, the researchers select a supported PDF-capable LLM provider and model, enter the corresponding API key, and the model converts the document into text. The conversion prompt instructs the LLM to preserve the document's wording and order, transcribe tables, and insert descriptions of figures, charts, and other images inline. CAT displays the converted text for review and editing before the researcher uses it as part of the experimental context. The provider and model used for PDF conversion do not have to be the same as those used for the subsequent coding run.

2.5 Models and Aggregation

Because LLM outputs are inherently stochastic, even if the temperature is set to zero, CAT will generally *not* produce the same output if it is run more than once using identical inputs. To reduce variability and limit the impact of outliers, researchers often use multiple models and replicate the coding multiple times with each model. The results are then aggregated into a single coding using either the mode or average of the multiple codings. Baranski et al. (2026) present theoretical and empirical analysis showing that use of multiple models and replications reduces idiosyncratic noise, with use of multiple models being more effective than use of multiple replications. They stress that the use of multiple replications cannot eliminate model-specific biases. CAT easily implements use of multiple models and multiple replications per model.

The researchers configure one or more models by selecting an LLM provider and model and entering the corresponding API key for each. CAT currently supports models produced by OpenAI (ChatGPT), Google (Gemini), DeepSeek, Anthropic (Claude), and xAI (Grok). Multiple models are supported for each provider; see the website for details. The first three providers were used in Baranski et al. (2026); CAT has subsequently expanded its provider support. Model-specific configurations allow the researchers to adjust the temperature, top-p, and maximum number of output tokens (if the selected model supports those parameters as input). Each configured model runs independently. The researchers also choose the number of replications per model. CAT retains the individual replication responses in addition to the aggregate coding, and researchers can download all these results.

2.6 Running the Coding Task and Output

Once the coding task has been fully configured, the researchers choose how to execute it. They can directly code their communication data by clicking on the *Run Coding* button. CAT first validates the selected API keys and models, and then performs the coding in the browser. Progress and results are displayed as communication episodes are processed. After completion, CAT checks the returned values against the coding manual and identifies failed or out-of-range observations. The researchers can inspect the validation report and re-run just the affected communication episodes rather than the full dataset. A selective rerun replaces the previous value for each affected communication episode. CAT then maps the final communication episode-level values back to the source data. Here, mapping results back means assigning each communication episode-level code to every source row that belongs to that communication episode. The primary coded CSV file contains the same number and order of parsed source rows and the same original columns as the uploaded dataset, together with the appended final coding columns. When several source rows form one communication episode, those rows receive the same communication episode-level coded values. This file can be downloaded through the results section of CAT.

A separate optional CSV file contains one row per pre-processed communication episode and only the fields used to construct or describe the communication episode together with its final coding columns. When several models or runs are used, the researchers may separately download a ZIP archive containing CSV files for the aggregate and individual model-run results. Following a selective rerun, CAT replaces the earlier call-level and aggregate records for the affected communication episodes while retaining the records for all other communication episodes, so the updated detailed archive remains available for download.

Rather than proceeding directly to coding, researchers can click on the *Generate Package* button to create a ZIP archive containing a ready-to-run Python script, three exact input files named *source_rows.csv*, *episodes.csv*, and *row_map.csv*,[4] a README file with execution instructions, and a requirements text file listing the necessary Python dependencies. The script is configured to read the three exact filenames. By default, local execution produces a source-row CSV file containing the original columns and mapped coding values; the researchers can optionally request an additional communication episode-level CSV file. The

[4]The file *source_rows.csv* contains the uploaded original source dataset in CSV format; *episodes.csv* contains one row for each communication episode constructed from the source rows; and *row_map.csv* records which source rows belong to each communication episode, allowing the communication episode-level coding results to be assigned back to the appropriate source rows.

package does not include an API key. It is important to note that the current packaged script uses the first configured LLM and does not reproduce the multi-model execution available through *Run Coding*. [5]

3 A Guided Example

This section illustrates the CAT workflow using the constructed example provided in the guided tour on the website. The example contains six messages exchanged by three different participants across two sessions and two rounds. The example is intentionally small so that the relationship between source rows, communication episodes, and coded results can be followed directly. The classification results are illustrative and are not intended to establish model validity.

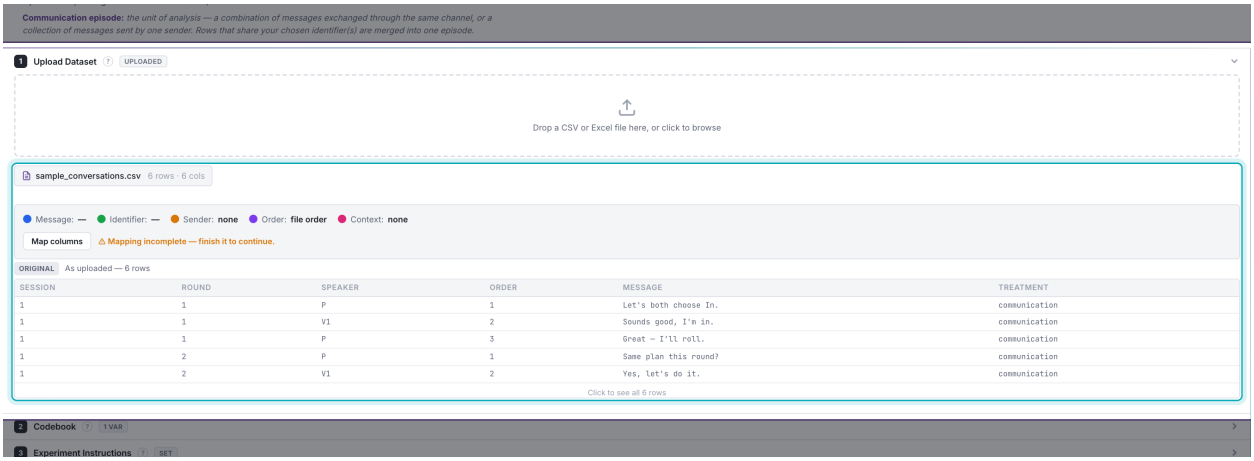

Figure 2: Upload your dataset

The workflow begins with uploading the source dataset as shown in Figure 2. When uploaded, CAT displays a preview and opens the *Map Columns* window. Figure 2 shows the uploaded file name and confirms that the example contains six source rows and six columns. The researchers then specify how the source columns should be used to construct the communication episodes submitted to the LLM.

[5]The source code for the CAT application, which generates these project-specific scripts, is separately available for inspection at `https://github.com/jeongk31/ssel_llm_tool`.

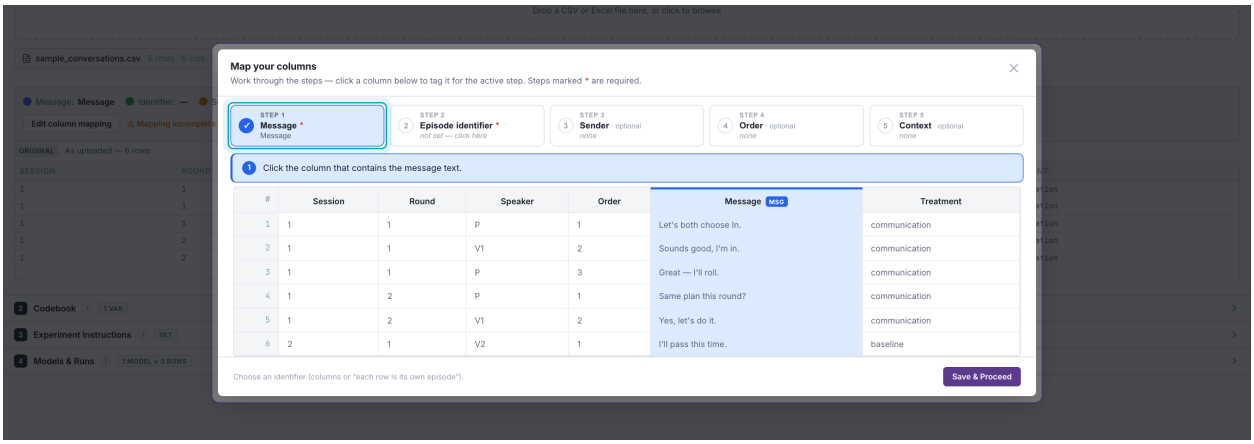

Figure 3: Map your columns - Message

The example first selects the *Message* column as the column containing the text to be coded. As shown in Figure 3, the *Message* column is highlighted in blue and marked with the *MSG* tag. This mapping is required.

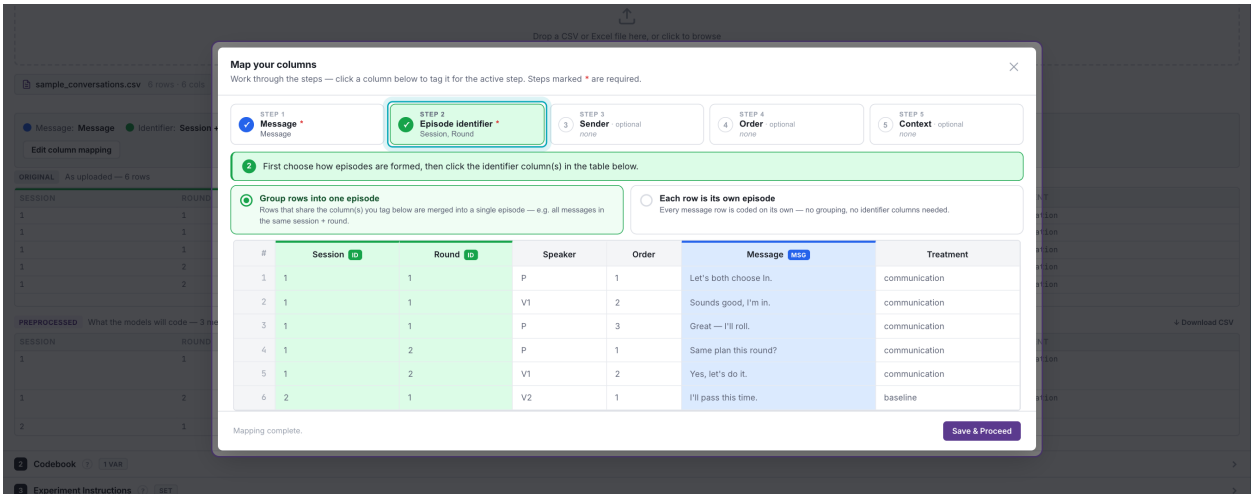

Figure 4: Map your columns - Episode identifier

The example then selects *Session* and *Round* columns jointly as the communication episode identifiers. Figure 4 shows both columns highlighted in green and marked with *ID* tags, indicating that CAT will use their values together to identify the communication episodes. CAT therefore combines rows that share the same values in both columns into communication episodes. The six source rows form three communication episodes in this case: the first contains three messages, the second contains two, and the last contains one. This step is required and important as it determines the unit of communication presented to the model.

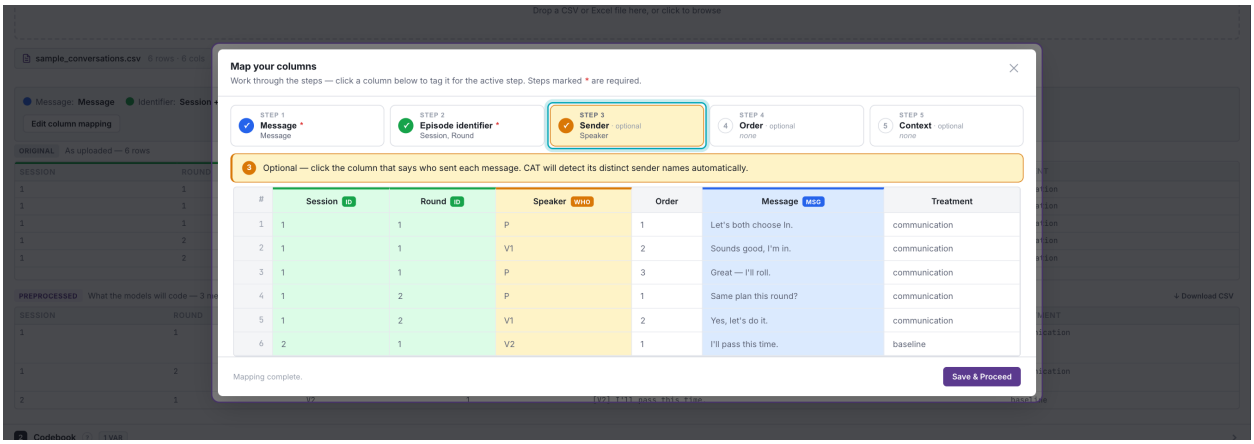

Figure 5: Map your columns - Sender

The *Speaker* column is selected as *Sender*. As shown in Figure 5, the column is highlighted in orange and marked with the *WHO* tag. CAT detects the three participant identifiers from this column: *P*, *V1*, and *V2*. Sender information allows the model to distinguish message sent by different subjects and is required only when a category is coded separately for each sender.

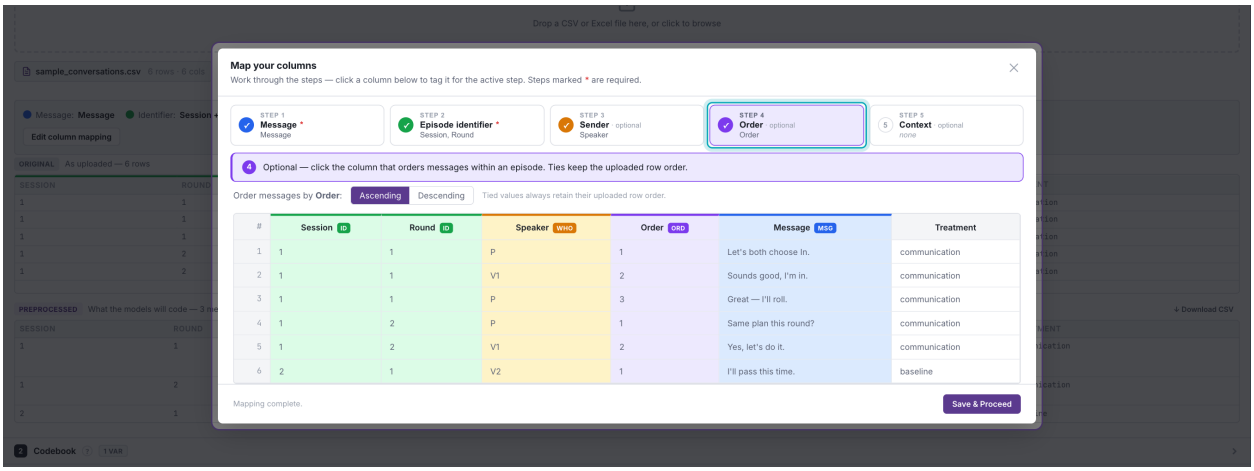

Figure 6: Map your columns - Order

The *Order* column is selected to arrange messages in *Ascending* order within each communication episode. Figure 6 shows the *Order* column highlighted in purple and marked with the *ORD* tag; it can also be seen that *Ascending* order is selected for the ordering method. Preserving this sequence is necessary when the interpretation of a message depends on preceding proposals, responses, refusals, etc. If multiple rows have the same order value, CAT retains their relative order in the uploaded dataset.

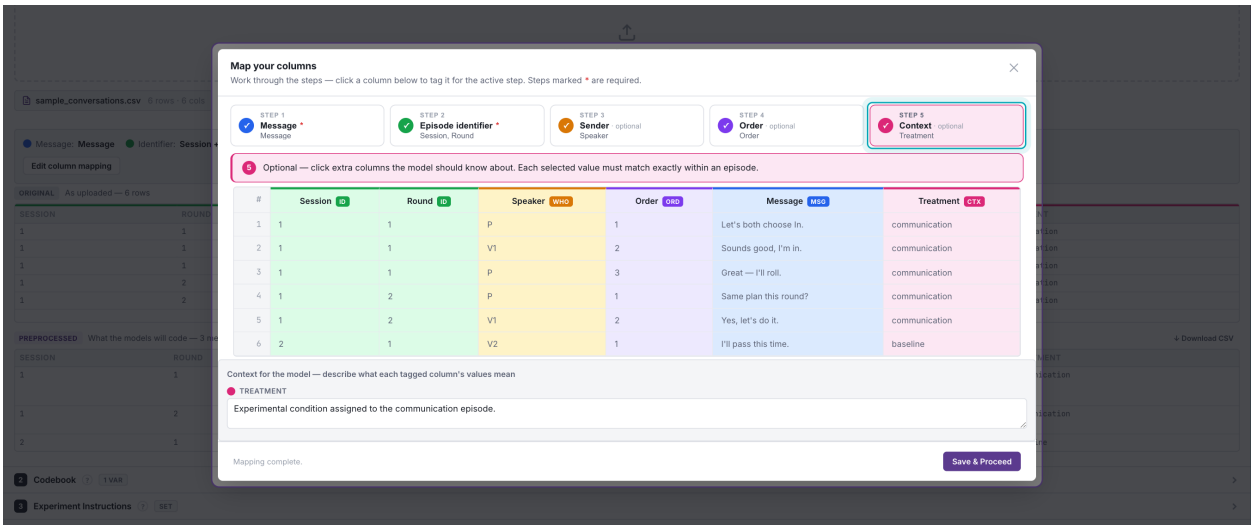

Figure 7: Map your columns - Context

The *Treatment* column is selected as *Context*. As shown in Figure 7, the column is highlighted in pink and marked with the *CTX* tag. The text box below the table describes *Treatment* as the experimental condition assigned to the communication episode. CAT supplies this information to the model alongside the communication episode text. Because a single *Context* value is attached to each communication episode, CAT requires the selected field to have exactly the same value across all source rows grouped into that communication episode. Any inconsistency must be corrected in the source dataset or the variable must be unselected as *Context* before proceeding.

Figure 8: Empty messages

After the column mappings are saved, CAT constructs and displays the three communication episode-level coding units. The example then specifies how communication episodes with an empty *Message* column (i.e., no messages were sent) should be handled. As seen in Figure 8, the example uses *Ignore*, under which CAT skips the model call for such empty communication episodes but retains the corresponding source rows in the primary output

with blank coding fields (i.e., missing observations). Directly below it, the Codebook summary lists the variables to be coded and provides access to the full editor.

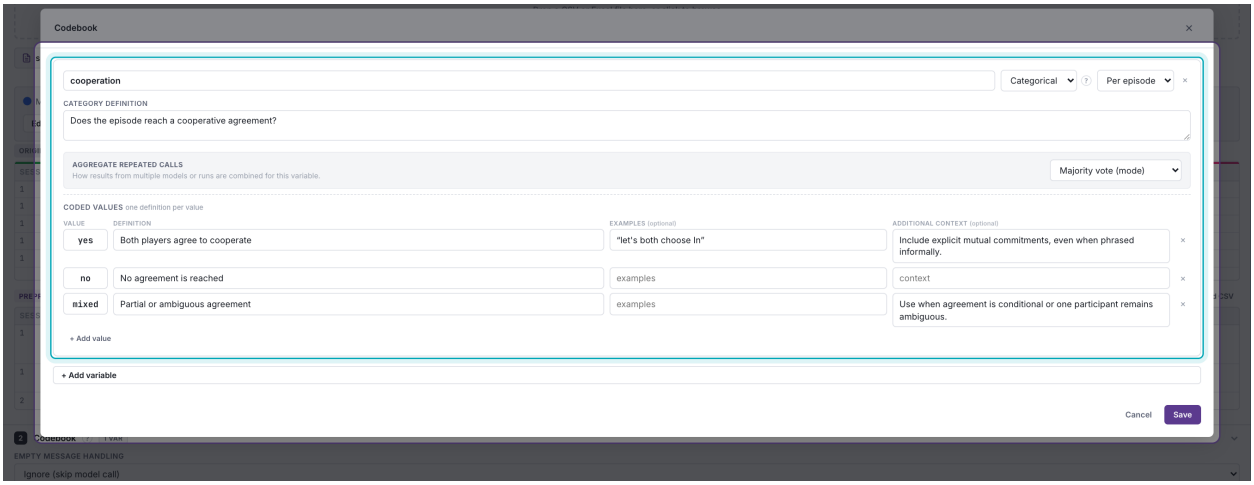

Figure 9: Variable card

The coding manual for the example contains one categorical variable labeled *cooperation*. It is coded once per communication episode and asks whether the participants reach a cooperative agreement during the communication episode. The permitted values are *yes*, *no*, and *mixed*. Each value is defined separately, with examples and additional context to clarify how particular statements or ambiguous cases should be classified. Figure 9 shows these definitions in the *Coded Values* area of the variable card, while the *Aggregate Repeated Calls* menu shows that *Majority Vote (mode)* has been selected. The example uses majority vote to aggregate repeated responses for this variable. Aggregation is specified separately for each codebook variable.

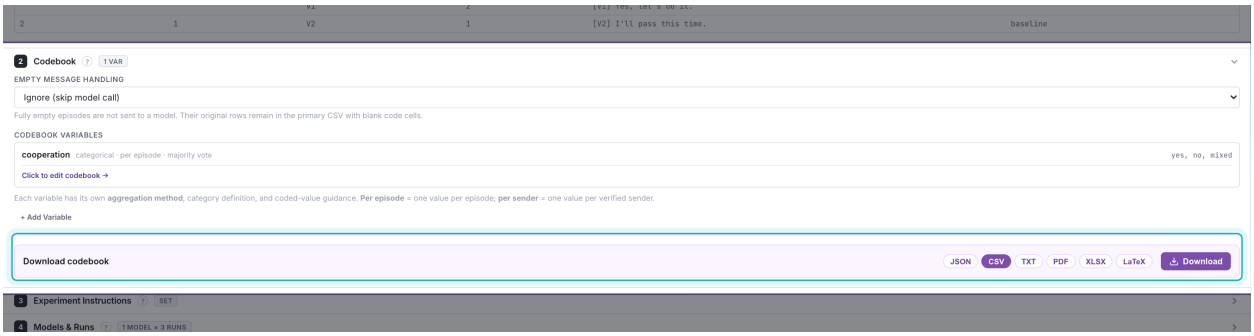

Figure 10: Download codebook

Once the coding manual has been saved, researchers can download it for review, reuse, or inclusion in the project's documentation. As can be seen in Figure 10 CAT supports the

following file types: JSON, CSV, TXT, PDF, XLSX, and LaTeX.

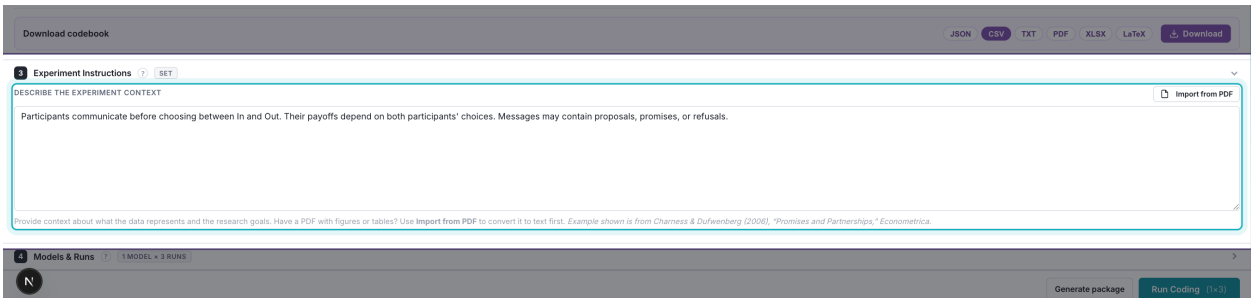

Figure 11: Experimental instructions

The experimental instructions explain the overall structure of the experiment and the role of communication in the experiment. This information helps the model interpret statements whose meaning depends on the rules or setup of the experiment. The instructions are supplied to the LLM as background information and are not themselves treated as messages to be coded. CAT also allows a PDF import. Figure 11 shows the instructions entered in the text box and the *Import from PDF* control immediately above it. The example instructions explain when participants communicate, what choices they make, and that their payoffs depend on both participants' choices.

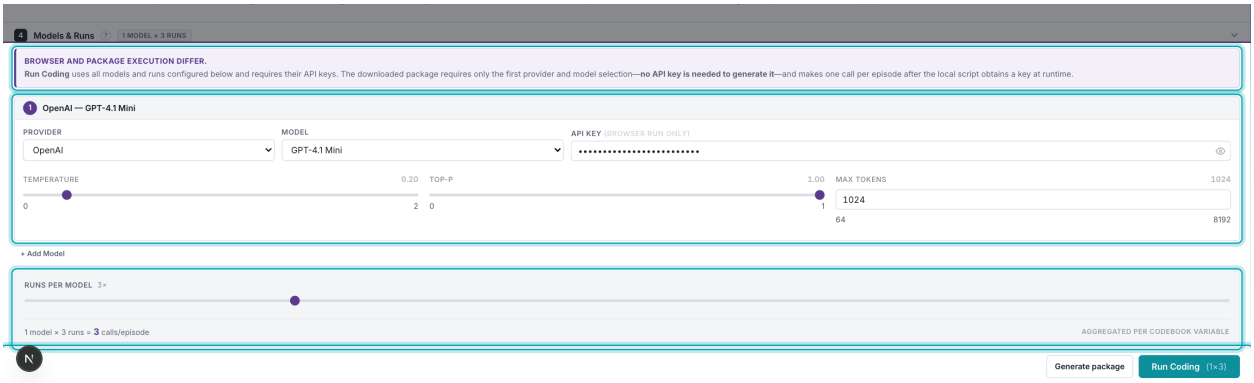

Figure 12: Models & API keys

As shown in Figure 12, the example configures OPENAI's GPT-4.1 Mini with a temperature of 0.2, a top-p value of 1, and a maximum output length of 1,024 tokens. *Temperature* controls response variability, with lower values generally producing more consistent responses and higher values allowing greater variation; the example uses 0.2 for relatively consistent responses. *Top-p* controls the range of responses the model may consider: a value near 0 restricts it to only the most likely options, while 1 allows the full range. We set the value to 1

so that the variation is controlled through temperature alone. The 1,024 token limit provides sufficient space for the coding responses without allowing unnecessarily long outputs. The number of calls is set to three, so the selected model does the coding exercise three times. CAT retains the individual responses and applies the majority-vote rule specified for the *cooperation* variable.

Researchers can run the configured task in the browser or generate a package for local execution. The *Run Coding* and *Generate Package* buttons are shown at the bottom of Figure 12. Browser execution supports the configured models and repeated calls. The generated package contains all files required to fully run the coding task, but is currently limited to one LLM and call. No API key is included in the downloaded package.

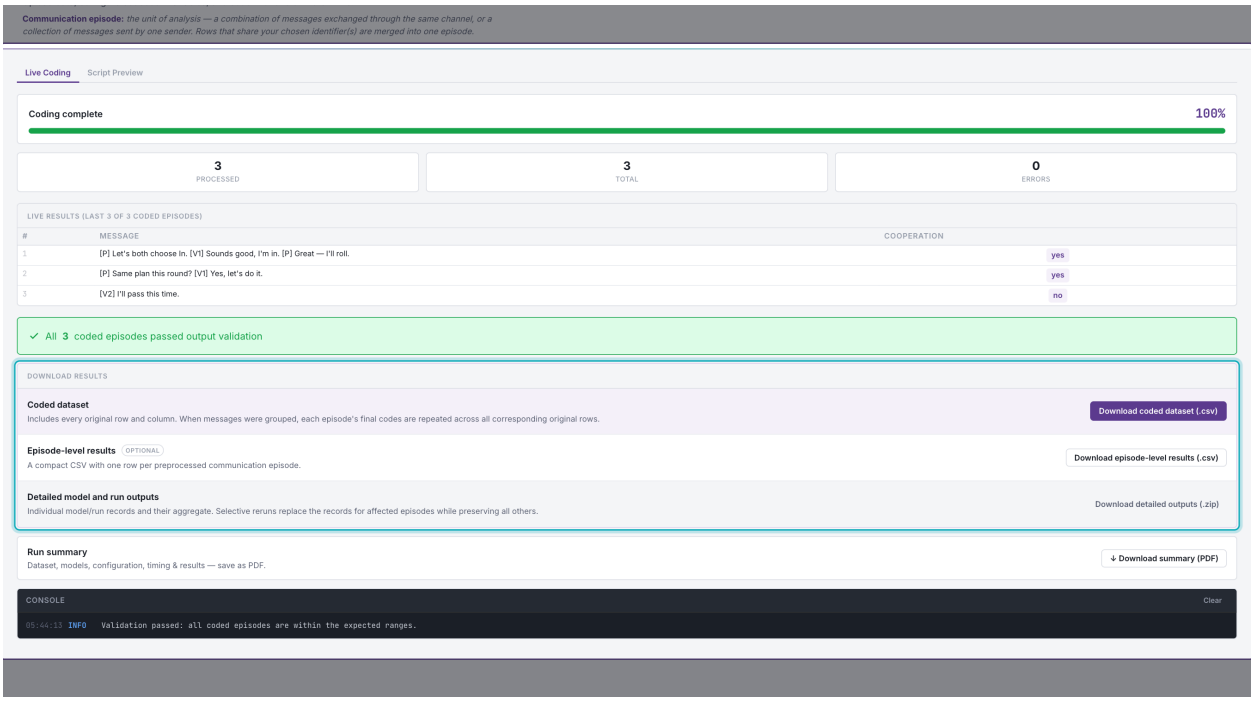

Figure 13: Review and download results

The example returns three communication episode-level classifications. Figure 13 shows one result row for each communication episode and reports that all three outputs passed CAT's basic checks. If a communication episode instead contained a failed call or an invalid value, the researchers can inspect the reported issue and selectively rerun just that communication episode.

The primary download is a CSV file with the same six rows and original columns as the uploaded dataset. CAT maps each final communication episode-level classification back

to every source row belonging to that communication episode. The three rows in the first communication episode therefore receive the same code, as do the two rows in the second communication episode. Researchers may optionally download a communication episode-level CSV containing one row for each preprocessed communication episode and a detailed archive containing both the aggregate and individual model-run records. These three corresponding download options appear beneath the results table in Figure 13.

4 Technical Aspects of Data Handling and Procedures

4.1 Data Flow

CAT consists of a browser client and an application server. The browser supports file selection, column mapping, codebook editing, model configuration, progress monitoring, basic output checks, and review of results. The server rereads and preprocesses the uploaded file for execution, checks the consistency of selected communication episode-level *Context* fields, constructs provider requests, parses returned JSON, aggregates successfully parsed calls, maps final communication episode values back to the source rows, and generates downloadable CSV datasets and other requested artifacts..

During the execution, the server sends the experimental instructions, codebook, communication episode texts, and other input data to the external LLM provider chosen by the researchers. API keys pass through the CAT server so it can validate credentials and make the requested calls. CAT does not persist API keys in any form, including the generated packages.

Uploaded datasets and generated results are written to temporary server-side working directories. The interface requests deletion when the current dataset is replaced or the project is reset, and an hourly cleanup process removes CAT temporary directories older than 24 hours as a backstop. Closing a browser tab does not itself request immediate deletion. The browser also retains the uploaded table preview, mapping, experimental instructions, and codebook in local storage so a setup can be restored after a refresh; this copy remains until the project or browser storage is cleared.

Operational analytics are stored separately in PostgreSQL. The current implementation records a persistent browser session identifier, IP address and IP-derived location, browser and referral metadata, selected provider and model names, and counts describing the config-

ured task. Dataset text and API keys are not saved at any point. Contact-form submissions are stored in a separate table.

4.2 Procedural Reproducibility

Reproducibility in LLM-assisted coding has at least three components. First is data construction, which consists of the source file, tagged columns, the uploaded-row tie rule, contextual fields and their within-episode consistency, and treatment of empty messages. Second is coding specification: experimental instructions, category definitions, permitted values, examples, coding scope, the automatically detected and verified sender list where applicable, and category-specific aggregation rules. Third is the model and execution configurations, consisting of provider, model, parameters, number of calls, and execution date. A final coded CSV is insufficient to reconstruct the coding exercise. Researchers should therefore retain the source and preprocessed CSV files, mapping and context-consistency decisions, configuration records, call-level outputs where applicable, aggregate results, and rerun history. These materials support procedural reproducibility, but due to the stochastic nature of LLMs, they cannot guarantee identical future classifications. Additionally, LLM outputs may vary across calls, and providers may update or retire models even when the reported model name and settings remain unchanged.

5 Discussion and Conclusion

The primary goal of this note has been to introduce the Communication Annotation Tool (CAT), an LLM-based tool for content analysis. CAT implements the methodology for LLM-based coding introduced in Baranski et al. (2026). We strongly recommend that readers of this note review that paper for a more general discussion of the use of LLM-based coding, and we request that users of CAT cite Baranski et al. (2026). CAT is intended for use with communication datasets from economic experiments, although there is nothing *per se* that prevents it being used for other content analysis tasks.[6]

This note focuses on the nuts and bolts of working with CAT. We describe how to use CAT to categorize communication from an experimental dataset, go through a step-by-step exam-

[6]Researchers using CAT for other content analysis tasks should be aware that the accuracy of LLM-based coding varies across tasks. Existing evidence supports the claim that LLM-based coding is reliable for communication data from economic experiments, but users should establish reliability for other settings. See Baranski et al. (2026) for a full discussion of this issue.

ple with a small dataset, and discuss the technical details of implementing CAT. Additional documentation is included on the website housing CAT (https://ssel.chatkit.nyuad.nyu.edu/). As CAT evolves, updated documentation (including updated versions of this note) will be posted on the website. We doubt most readers will need to know the technical details of how CAT works, but readers should take advantage of CAT's ability to easily implement LLM-based coding *while adhering to established methodological standards*.

In line with the preceding comment, some cautionary notes are in order. CAT structures a coding procedure; it does not establish validity of the coding scheme. Its automated checks identify parsing failures and responses that do not conform to the coding manual, but CAT cannot determine whether the coding scheme measures the intended concepts and/or misses important components of the communication data. Put simply, CAT implements the coding scheme designed by the researchers verbatim. If the researchers miss important features of the communication data, correcting their mistake is beyond what CAT is designed to do. Readers who are interested in having LLMs design the coding scheme are directed to Cooper et al. (2026). Performance of LLM-based coding is variable, most often due to a lack of relevant context in the prompts. Researchers are therefore recommended to validate each coding task against an appropriate human-coded sample (see Baranski et al. (2026) for discussion of this issue). Researchers should always document the full coding process, including how the sample was selected, how agreement was assessed, and how disagreements were resolved.

Researchers need to be aware that coding exercises, including those implemented by CAT, do not somehow capture a ground truth. Communication is often inherently ambiguous, making it impossible to claim that any ground truth exists. Even if ground truth does exist, any coder (human or LLM) will provide an inherently biased view of the data. Aggregating across coders can reduce these biases, but it is impossible to completely eliminate them. Researchers should view coding output as variables with measurement error.

Use of CAT has important data-governance implications. The experimental instructions, communication episodes, and selected contextual information are transmitted to the LLM provider chosen by the researchers. To restore work after the browser is reopened, CAT retains the current dataset locally in that browser. The server uses a temporary working copy, and the application records the usage and request metadata described in the data-handling section. Researchers remain responsible for determining whether these operations comply with participant consent, institutional review requirements, data-use agreements,

and any other applicable law or policy. Sensitive data may require de-identification or an institutionally approved provider.

REFERENCES

Baranski, A., Cooper, D. J., and Lee, J. K. (2026). Are LLMs reliable coders of communication content in economic experiments? `http://hdl.handle.net/2451/75820`.

Brandts, J., Cooper, D. J., and Rott, C. (2019). Communication in laboratory experiments. *Handbook of Research Methods and Applications in Experimental Economics*, 401.

Cooper, D. J., McElvain, C., and Qi, S. (2026). Category generation, categorization, and replication: A complete method of llm-based content analysis. Working paper.

A Design Principles

CAT is designed to preserve researcher control over the decisions that define a coding task. The application does not infer the appropriate unit of analysis, coding categories, or substantive interpretation from the uploaded data. Instead, the researchers explicitly determines how messages are grouped into communication episodes (i.e., a series of messages which will be coded as a unit), which contextual information is supplied to the LLM, what each category measures and the outputs permitted, and how repeated LLM codings are aggregated into a final coded dataset (e.g., modal response, median, mean, etc.).

A second principle is to maintain a clear connection between the source data and the communication episode units submitted to the LLM for ease of data management and analysis. Communication data from experiments are often stored with one message per row, whereas the substantively relevant unit may contain several messages from the same session, round within a session, group of subjects paired together, or participant. CAT therefore constructs a separate, preprocessed communication episode-level dataset for coding while retaining a mapping to the original source rows. After coding, communication episode-level values are assigned back to every source row belonging to the corresponding communication episode. Researchers can consequently analyze the coded variables alongside the original data without manually reconstructing this relationship mapping.

Third, CAT is intended to make a coding workflow accessible to researchers with different levels of programming experience. The browser interface supports configuration, execution, validation, and export without requiring the researchers to write any code. The generated local-execution package exposes the corresponding data-processing and coding procedure for researchers who wish to inspect or modify it. These features improve the transparency and portability of the procedure.

A final principle is that of reproducibility: CAT produces an output documenting all the options used for the coding of communication as well as the instructions for coding (code book). Therefore, researchers can apply the same procedures in subsequent analyses or different datasets, and easily report them in their articles.

B Technical Architecture and Deployment

B.1 Web Architecture

CAT uses a client-server architecture. The browser interface is implemented in Next.js and React and is responsible for dataset selection, column mapping, codebook editing, model configuration, progress monitoring, and review of results. A FastAPI server performs file parsing and preprocessing, constructs requests to external LLM providers, parses and aggregates their responses, and prepares result files and local-execution packages. Pandas is used for tabular transformations that construct communication episodes and map their final codes back to the corresponding source rows.

The browser and server communicate over HTTPS. During browser execution, the server streams progress and results to the browser while it processes the configured model calls. Requests containing the experimental instructions, codebook, communication episodes, and selected contextual information are sent from the application server to the external LLM providers selected by the researchers. PostgreSQL is used separately for the limited operational metadata and administrative records described in Section 4.1; communication data and API keys are not stored in the database.

B.2 Deployment and Access

The production application is deployed on a university-hosted Red Hat Enterprise Linux server.Nginx terminates HTTPS connections and acts as a reverse proxy for the frontend and backend services, which are managed as systemd services. PostgreSQL provides the application's persistent metadata store. Researchers using the hosted interface need only a web browser and credentials for their selected LLM providers; no local installation is required.

The CAT codebase is version-controlled in a public GitHub repository. Deployment configuration and server administration remain separate from the browser workflow presented to researchers.