Paper PP10

Harnessing Large Language Models: Enhancing Test Data Quality and Efficiency in Clinical Trial Set-Up

Rita Pecuch, Takeda, Ann Arbor, United States
Varun Jagadeesh, Takeda, Boston, United States

ABSTRACT

Generating and entering test data into electronic data capture (EDC) systems is a critical yet laborious task, essential for validating study designs and numerous downstream systems and programs. We previously leveraged open-source technology to develop a tool to significantly ease the time-intensive effort of manual test data entry and notable efficiency gains have been achieved since its release; however, it falls short in effectively covering the full spectrum of test cases necessary for clinical system set-up and programming. To overcome these challenges, Takeda has sought to build on this innovative software solution to harness the power of large language models (LLMs) to expand the range of covered testing scenarios. During the development process, strategies and best practices for maximizing LLM potential in the clinical test data context have been established, along with valuable lessons learned along the way. Exciting potential future directions have been identified to further reduce clinical trial system set-up times.

INTRODUCTION

Electronic data capture (EDC) systems are used to collect and store large volumes of data for a clinical trial. Entering test data into EDC systems prior to first subject in (FSI) is a critical task to validate various aspects of each trial database design before using it to collect production trial data, as well as validating several downstream systems and programs that consume the data. When done manually, this is a labor-intensive task, especially for the validation of certain downstream programs which may require a complete set of test subject data, meaning all forms for all visits need to be filled out, and ideally all fields on each form. For many EDC trial builds, this is further complicated by the fact that visits must be added sequentially, meaning that if test data were only needed for a form occurring at one of the last visits, then all preceding visits would need to be added to the test subject casebook first. According to early testimonials from several of our clinical programmers, for large Phase III studies this task can take several hours, or a few business days when balanced with other work, which is the more common scenario.

To address this problem, we previously developed a Python application capable of generating a full set of subject test data and entering it into Veeva CDMS EDC for any of Takeda's trials. This application generates test data points in accordance with the guidelines in the Study Design Specifications (SDS) document, which summarizes all EDC trial design elements (for example, required forms and fields) in tabular format, and enters the data directly into EDC forms via an application programming interface (API) connection [1]. Besides downloading the SDS document from the EDC system and uploading it to the Python application, this is a completely automated process. However, while rules-based data generation is useful for some clinical programmers and testers, others need test data that mirrors production trial data as closely as possible or includes highly specific data points to validate trial design components such as edit checks. This paper explains why we chose to leverage large language models (LLMs) to incrementally close these gaps, summarizes progress to date, and outlines lessons learned and how they may be applied to address additional gaps in the future.

OPEN-SOURCE TECHNOLOGY FOR TEST DATA ENTRY AUTOMATION

To automate the data entry component of test data strategy, we previously developed a Python application to automate the generation and entry of complete subject test data into Veeva CDMS EDC as shown in Figure 1 [1].

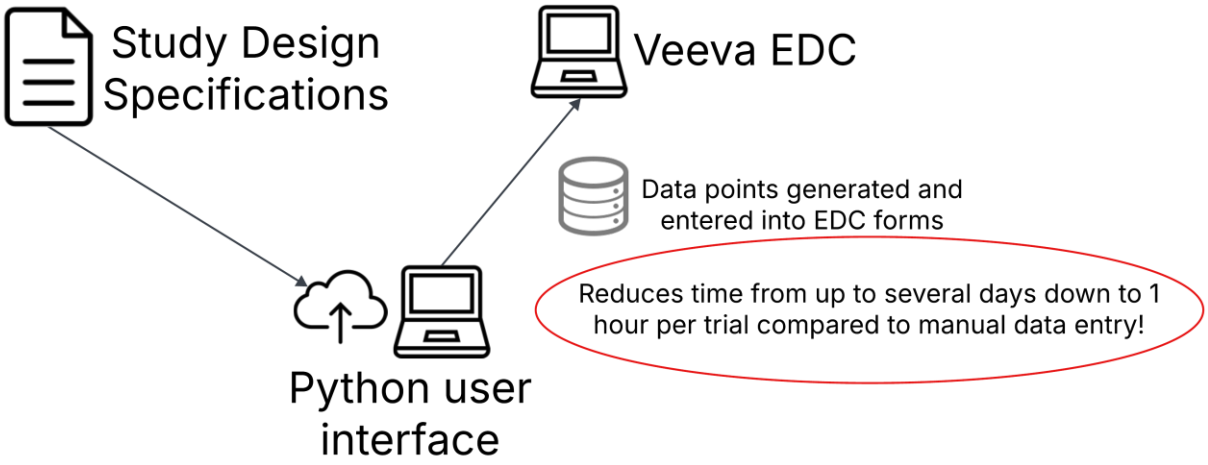

Figure 1: Architecture for Veeva test data entry application. The user uploads the Study Design Specifications document (SDS) for their trial, which was exported from Veeva CDMS EDC, to the user interface. When the user clicks a button, test data points are generated for the trial using rules-based Python logic and entered into Veeva CDMS EDC forms via the Veeva CDMS API. This application has introduced significant time savings into Takeda's test data entry process.

A similar application could theoretically be created for any other EDC system with an API that has endpoints for creating subjects, adding visits, adding forms, adding form data, and submitting forms. We additionally recommend ensuring that the API has endpoints available for retrieving the trials and sites that a user has access to in the EDC test environment, which will allow including a check in the application to inform the user if they need to first obtain needed access. To comply with our user access management standard operating procedures (SOPs) for EDC access, we chose to prompt users to enter their EDC credentials in the application interface rather than use a shared service account, ensuring that only users granted API access by an authorized team can initiate test data entry.

A few challenges encountered when developing this application were handling limitations on accepted visit dates for some trials and handling conditional forms. For many trials, the system enforces planned date ranges that require a visit to occur within a specified number of days of another visit (for example, the previous visit); if a date outside that range is used, the API returns an error message. However, the Veeva CDMS API helpfully provides the accepted date range in the error message, and we opted to use pattern matching to extract the first date of the accepted date range and use to re-try setting the visit date. Many trials additionally configure certain forms to only be added to a visit when a predefined condition is met. To meet the needs of current stakeholders, we opted to use an API call to forcefully add every form, including these dynamically triggered forms, even when the triggering condition was not met. This strategy ensures that all subject forms contain test data and can be shared with stakeholders, including testers of data transformation pipelines and clinical programmers who produce trial reports for data review, safety monitoring, and operational oversight.

Since its release in April 2024, this application has been used in nearly all new trials that Takeda has built within Veeva CDMS EDC, as well as to support programming updates to reports for existing trials. As of January 2026, this has amounted to 40 trials. According to early feedback, clinical programmers report that the application eliminates most of the manual effort required to enter test data into the EDC system to begin programming; if specific data points are needed on particular forms, they can update those values manually after the bulk automated data entry is complete. Additionally, when programming listings in Veeva CDB (one of the data review tools used at Takeda), if a field has not yet been entered for a subject, it does not appear as a column in data extracts (rather than appearing as a blank value). Usage of our application one time, which takes only a few minutes of manual effort, eliminates this issue. Our clinical programmers have stated that this application consistently saves them 1 week of effort per batch of 20 listings, which realistically amounts to a few weeks of time savings per trial. Our statistical programmers have reported that the application's ease of use and enterprise-wide availability enables rapid generation of large numbers of test subjects, allowing programming to begin earlier. According to an early testimonial, they typically target about 30 test subjects to adequately test statistical analysis code, which we have been told would require roughly 60 hours of work per trial without an automated solution.

For our testers of data transformation pipelines, test data is required for all fields on all forms that are involved in the specific data transformations. The biggest benefit that has been described to us for this team is not needing to plan resource availability and prioritization of data entry requests, and there are typically multiple per trial as database

updates are made. Any member of their team can use the application at any time, as only a few minutes of manual effort is required and the test data is then available in the EDC system in around an hour.

LEVERAGING LARGE LANGUAGE MODEL FOR TEST DATA GENERATION

LARGE LANGUAGE MODEL SELECTION

Despite the significant efficiency gains reported so far in Takeda's test data entry process, a rules-based approach to generating test data points can cover only a limited set of testing scenarios. The variety in clinical trial designs—and, in turn, EDC trial builds—can quickly lead to an endless cycle of adding conditional rules. For example, most EDC builds include a designated screening event with forms used to assess a subject's eligibility for the trial, but the screening visit is not named consistently, and some trials include multiple screening visits. While pattern matching could ideally be used if all screening visits at had the same root (i.e. "screen"), we find that some trials use less descriptive visit names such as "Visit 1", "Visit 2", etc. However, given a visit list, the forms expected at each visit, and brief context about a subject status (e.g., screen failure, early termination), we observed that large language models (LLMs) identified screening visits with extremely high accuracy—consistently achieving 90–100% accuracy across 13 studies that we considered representative of the diversity in our current trial builds, spanning multiple therapeutic areas and phases (Table 1). It was this observation that led us to believe that incorporation of an LLM into our arsenal of tools for this application had great potential to provide value and open the doors for much greater test scenario coverage.

A language model is a statistical model trained to predict sequences of tokens in human language, enabling it to generate and transform text. Large language models (LLMs) build on this foundation by training on extremely large and diverse datasets and, when instruction-tuned, can follow human directions and handle complex, context-dependent inputs. Because our application must interpret highly variable trial design text and generate realistic, context-appropriate test data—rather than only classify or extract information—LLMs offer clear advantages over more traditional natural language processing (NLP) approaches, which are more narrowly trained and have limited capacity across diverse clinical trial builds. As a result, LLMs provide a practical path forward for meeting these evolving needs.

To select a suitable LLM to incorporate into this application, we partnered with our enterprise technology, information security, and legal teams to assess the risk of the proposed use case. This rigorous review was necessary because, although we intended to use a pre-trained model that would not continue learning from our prompts, LLMs are still classified as artificial intelligence (AI), and their use is tightly governed at Takeda. After this evaluation, we were granted access to a private endpoint for a Takeda-approved LLM that does not learn from input prompts.

APPLICATION ARCHITECTURE

Rather than relying on ad hoc chatbot interactions, we programmed a set of thoroughly tested prompts directly into the application and the LLM is invoked through an API. This approach standardizes how the model is prompted and maximizes output consistency. Our most current release of the application, which is currently undergoing formal validation as of January 2026, incorporates three prompts that were thoroughly tested on the 13 studies that span several therapeutic areas and phases described in Table 1. We opted to begin with relatively simple prompts that we expected to produce highly consistent, accurate outputs and deliver some value to current stakeholders, while giving the development team time to build confidence working with LLMs before pursuing more complex use cases, such as generating test data tailored to edit checks or programmed reports.

Therapeutic Area	Phase(s)
Gastrointestinal	IIb, III
Neuroscience	IIa, III
Oncology	I/II, III, IV
Plasma-derived therapies	II, III
Rare genetics & hematology	III
Vaccine	III, IV

Table 1: Breakdown of trial designs used for prompt engineering and testing

The application consists of two buttons. When the user clicks the first button, the application triggers an initial prompt that provides the LLM with a list of clinical trial visits extracted from the SDS document and asks it to select the visits required for a test subject with a specified subject status. The prompt is slightly different based on the subject status

selected by the user and can currently handle screen failure and early termination statuses. The output of the first prompt, together with the subject screening date entered by the user, is then passed into a second prompt, which generates realistic dates for each subsequent visit based on the visit names and the screening date.

The rationale behind including a date selection prompt lies in the options which Veeva CDMS EDC provides for setting limits on date ranges. For many trials, the system enforces planned date ranges that require a visit to occur within a specified number of days of another visit (for example, the previous visit). Under this scenario, the API can be used to set realistic visit dates by first attempting to assign a date, parsing the expected date from the resulting error message, and then retrying with the corrected value. On the contrary, under scenarios where no such planned ranges are configured, it is difficult to deduce scalable logic that works for a variety of visit names to generate a realistic schedule of dates. However, this is a task that LLMs can handle with ease, and by doing so we ensure the downstream programmers are not working with unrealistically spaced or overlapping visit dates, as our existing rules-based test data generation logic uses visit dates to populate many non-historical date fields on forms. Our clinical programmers have informed us this will save them additional time by eliminating the need to log into the EDC system manually to adjust dates for date-comparison listings and date-based visualizations.

Although we achieved very high accuracy during early prompt engineering for our visit and date selection prompts (see the next section for how accuracy was measured)—including 100% accuracy for date selection and over 90% accuracy for visit selection across 13 representative studies—LLMs are probabilistic models and will almost always produce some variation in their responses. Because users of this application have the domain knowledge to judge which visits a screen failure or early termination subject should realistically complete, we allow them to re-click the visit generation button i the selected visits are not deemed appropriate, with the expectation that a subsequent run may produce a better result. This strategy also enforces the key concept of human-in-the-loop, which is in accordance with Takeda's recommended AI best practices, and is visualized in Figure 2.

The third prompt in this release generates test data for screening disposition, inclusion/exclusion criteria, end-of-treatment, and end-of-study forms. We intentionally limited the scope to this subset of forms because this is our team's first integration of an LLM into an application with high operational importance, and we wanted sufficient time to experiment with different approaches and closely monitor performance before extending the functionality to all forms. Formatting checks, described in more detail in the next section, are built into the application to ensure outputs are compatible with the existing data-entry code; if a check fails, the user is shown an error message. The user may then re-try, again enforcing the concept of human-in-the-loop. This strategy is visualized in Figure 3.

The subset of forms was chosen given the significance of these forms in differentiating a subject from a screen failure, complete, or early termination status. Additionally, the LLM is only used to generate data for these forms for screen failure and early termination subjects. For all other forms as well as subjects with a complete status, the existing rules-based logic is still used to generate test data points. It was important to preserve the current test data generation logic for complete subjects, as certain current stakeholders rely on the fact that all fields are filled in.

The below diagram depicts the application architecture and flow of events. Python remains the primary technology used in this application, and the LLM is simply an additional tool used for specific steps where it adds value.

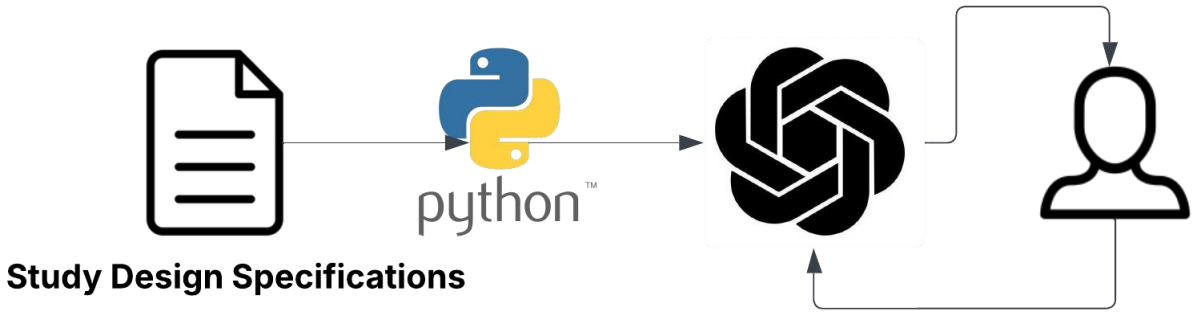

Figure 2: Visit List Generation. The user uploads the Study Design Specifications document (SDS) for their trial, which was exported from Veeva CDMS EDC, to the user interface and selects a test subject status. When the user clicks a button, Python code is used to extract a list of visits from the SDS and send an API call to the large language model (LLM) to prompt selection of visits applicable to the selected status. The user evaluates the accuracy of the list of selected visits and re-clicks the button if applicable.

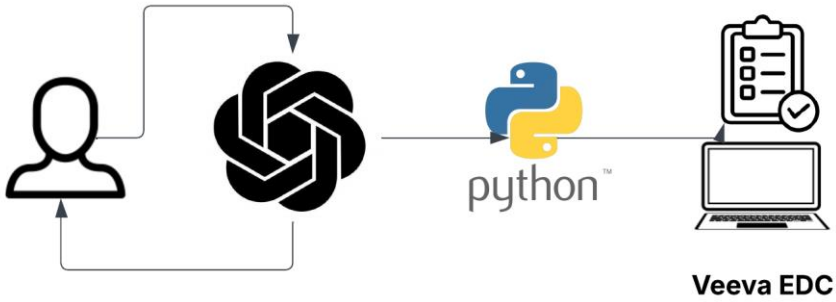

Figure 3: Test Data Generation and Entry. Once satisfied with the list of selected visits as described in Figure 2, the user clicks a second button, which sends an API call to the LLM to prompt date selection and test data generation for applicable forms. If any formatting checks fail, an error message is displayed and the user can re-click the button to re-prompt the LLM. If all formatting checks pass, Python code is used to generate test data for remaining forms and all generated test data is entered into Veeva EDC forms using the Veeva CDMS API.

PROMPT ENGINEERING STRATEGY AND LESSONS LEARNED

Before integrating the LLM prompts into the application, our first step was to engineer and refine them to consistently produce the desired outputs for each prompt described in the previous section. To easily track which versions of the prompts produced which outputs, we stored the prompt code and a record of the outputs in a GitHub repository. For each version of each prompt, testing was performed on the 13 trials described in Table 1. Each trial was tested 5 times on each subject status to check for variation in responses. The outputs from all iterations were written to a CSV file, which we could easily review for accuracy and identify inconsistencies to address through prompt refinement. This process was performed 17 times before accuracy converged, and we were then comfortable incorporating the prompts into the application.

Within the Python code we developed for sending prompts to the LLM, we programmed a few checks on the output of each prompt that we deemed absolutely necessary for accuracy and compatibility with the existing data entry logic of the application. Table 2 summarizes the checks that were included. Failure of any checks, which occurred less than 5% of the time during our testing, would throw an error, and in practice (within the application) this is not a problem because the user would see this message and be asked to try again.

Prompt	Check Type	Check Description
Visit Selection	Format	Ensure verbatim output from LLM can be evaluated as a Python list
Visit Selection	Controlled Terminology	Ensure all selections are present in SDS
Date Selection	Format	Ensure verbatim output from LLM can be evaluated as a Python list
Date Selection	Date Format	Ensure all selected dates are in expected date format
Date Selection	Length	Ensure number of dates selected matches the number of visits selected (excluding log events which do not have an associated date)
Data Generation	Format	Ensure verbatim output from LLM can be evaluated as a Python list of dictionaries
Data Generation	Format	Ensure each Python dictionary contains necessary key names

Data Generation	Controlled Terminology	Ensure generated data points adhere to controlled terminology if specified in SDS
Data Generation	Date Format	Ensure generated data points for date fields are in expected date format

Table 2: Checks performed on large language model (LLM) output

The most common failure that we observed for any of the checks was failure of the format check (described in Table 2) for the data generation prompt, and an example of how this would be displayed to the user is shown in Figure 4. In this example, instead of returning a list of Python dictionaries containing the test data, the LLM wrapped the output in backticks—likely because many chat interfaces render Markdown and backticks are commonly used to display code blocks. However, this cannot be cleanly parsed in Python, hence throwing an error and asking the user to try again.

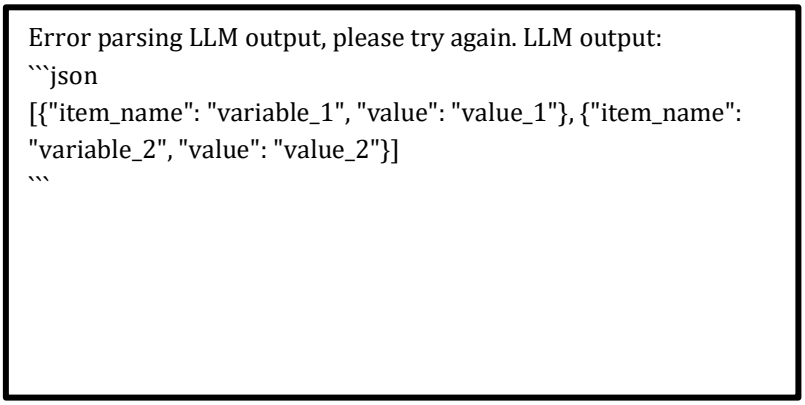

Figure 4: Example of LLM output check failure

When assessing outputs produced by each prompt that passed all checks, the results were grouped by trial and by subject status for some automated analyses. For the visit selection prompt, we assessed for consistency. Any trial/subject status combination not producing 100% consistency in responses was analyzed further to determine ways in which the prompt could be refined. For the date selection prompt, we assessed that the dates were in chronological order. We found that all trial/subject status combinations passed these assessments and additionally performed manual checks against visit names to verify that the spacing was as expected. For the data generation prompt, we manually assessed the output for accuracy. These accuracy criteria were slightly different for each form and are summarized in Table 3.

Prompt	**Accuracy Criteria**
Visit Selection	Results for the same trial and subject status should stay consistent across multiple iterations of prompting
Date Selection	Dates should be in chronological order
Date Selection	Dates should be spaced in a manner consistent with visit names (i.e. Week 1, Week 2)
Data Generation – Inclusion/Exclusion Criteria Forms	For a screen failure subject, test data should indicate that the subject did not meet all screening criteria and at least one criteria not met should be selected. For an early termination subject, test data should indicate that the subject met all screening criteria and no criteria not met should be selected.
Data Generation – Screening Disposition Forms	For a screen failure subject, test data should indicate that the subject did not enroll into the study and indicate a reason why. For an early termination subject, test data should indicate that the subject enrolled into the study and not fill out any fields that ask for reasons why not enrolled.
Data Generation – End of Treatment Forms	Not executed for a screen failure subject. For an early termination subject, test data should indicate that the subject did

	not complete treatment and indicate a reason why.
Data Generation – End of Study Forms	Not executed for a screen failure subject. For an early termination subject, test data should indicate that the subject did not complete the trial and indicate a reason why.
Data Generation	All fields should contain a value, unless expected to be blank based on subject status
Data Generation	All fields containing a value should adhere to expected data type as defined in the study design specifications

Table 3: Criteria used for evaluating accuracy of LLM output

A few main takeaways and lessons learned that helped us refine our prompts were utilizing system prompts and keeping programmed user prompts as concise as possible. System prompts provide context that guides how an LLM should behave, and we found that defining a role and specifying the required output format in the system prompt significantly improved output consistency. We found it was important to include all pertinent information in the user prompts, but that presenting it concisely also improved consistency by reducing the number of tokens the LLM had to process. Filtering and including only small chunks of the SDS in our prompts significantly improved the consistency in the LLM outputs we obtained compared to including the entire SDS or even an entire sheet within the file.

Finally, we found that most sources of inconsistency could be easily addressed by adding Python post-processing code that ran after the LLM response was generated but before we analyzed the results. For example, in several trials, the only inconsistency we observed during early prompt engineering was whether unscheduled visits were selected; these visits seemed to confuse the LLM and were included for screen-failure and early-termination subjects in some runs but omitted in others. For this reason, we opted to remove any selected unscheduled visits for screen failures and add one for early terminations. We recognize that these are assumptions that would not hold true across all subjects in a trial, but it was important that we established measures of accuracy that could realistically be achieved.

POTENTIAL FUTURE DIRECTIONS

Over the next few years, we plan to expand the capabilities of this application by working with new stakeholders and continuing to monitor and refine current capabilities. While the specific new capabilities will be prioritized and defined in collaboration with relevant business functions, potential next steps include supporting our EDC testing team by generating test data for EDC rules testing and further supporting clinical programmers by generating test data for report testing. We will discuss several approaches and collaborate to determine the best path forward for each new capability, whether that involves traditional LLM prompting, tool calling, leveraging historical data, or other NLP methods.

Because this application is already used across many trials and stakeholders have been highly engaged—providing consistent feedback—we are confident that any LLM performance degradation affecting the application will be reported to us quickly. The users directly see the results of the visit selection prompt, and results of the date selection and data generation prompts will ultimately be visible within the Veeva CDMS EDC system. Our current SOPs require quarterly LLM monitoring for low risk AI use cases, which is the classification that this application now falls under according to our enterprise technology teams. In quarters when we receive limited stakeholder feedback, we plan to analyze databased LLM outputs using the same approach as during prompt engineering and testing to detect any significant drops in accuracy that may need to be addressed.

CONCLUSION

EDC test data generation and entry is essential for validating trial database designs and downstream programming, but manual entry is labor-intensive and rules-based automation alone cannot cover the full diversity of trial designs or the highly specific scenarios needed for thorough testing. Building on Takeda's Python application that generates test data and enters it via the EDC API, we incorporated LLMs into targeted workflow steps to better interpret variable trial design text and produce context-appropriate outputs, while maintaining operational stability through rigorously tested prompts, formatting checks, and post-processing. We also established practical prompt engineering and evaluation practices to improve consistency and enable ongoing performance monitoring. Our promising results support expanding LLM-enabled capabilities to more complex use cases such as generating test data tailored to EDC edit check validation and downstream report and visualization testing, in close collaboration with stakeholders and under appropriate governance.

REFERENCES

1. Pecuch, R., Shetty, K., Jagadeesh, V. (2025). Automating Clinical Test Data Generation and Entry with Generative AI and Innovative Technology [Poster PP09]. Takeda. https://phuse.s3.eu-central-1.amazonaws.com/Archive/2025/CSS/EU/Utrecht/POS_PP09.pdf

ACKNOWLEDGMENTS

We would like to acknowledge the below teams and individuals at Takeda for their contributions to the development and maintenance of this application.

Other developers of the application: Rajkumar Poosarla, Sona Ashok Kumar, Keshari Gupta

Individuals and teams that have consistently provided valuable feedback and suggestions for the application:

- Data Configuration Engineering team, especially James Lin, Abhilash Sreenivas, and Kelsey Sheak
- Clinical Trial Reporting team, especially Allison Prey and Amber Hannah
- Testing, Innovation, and Data Enablement (TIDE) leadership team, especially Aviad Adlersberg, Aarti Sanadi, and Carol Wingate

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the authors at:

Author Name: Rita Pecuch
Company: Takeda
Email: rita.pecuch@takeda.com

Author Name: Varun Jagadeesh
Company: Takeda
Email: varun.jagadeesh@takeda.com