Paper ML16

Detecting Logical Errors in Clinical Programming by Applying ChatGPT

Illia Skliar, Intego Group, Kharkiv, Ukraine

ABSTRACT

The paper presents an approach for the semantic SAS-code review in the innovative field of clinical programming with the help of NLP ChatGPT® model. The primary goal is to detect logical inconsistencies or even mistakes between the implemented SAS code and rules, which are outlined in the unstructured but critical clinical trials documents such as Statistical Analysis Plan (SAP) or Protocol.

Unlike standard tools such as PROC COMPARE, which only focus on comparing output results, the integration of a GPT model through an API directly into the SAS environment is proposed. This approach specifically targets the validation of the logic implemented in the code fragments, for example, the assessment of Best Overall Response (BOR).

The key part of the method is a strictly formalized structure of the queries (prompt engineering), which allows to create sustainable interaction with the language model based on the input code and free unstructured text from documentation. To verify the validity of GPT responses, several approaches are suggested: manual validation, comparison with legacy code, and peer-review logic by colleagues.

Potential limitations of ChatGPT® context are considered, including the possibility of generating inaccurate conclusions (hallucinations), limitations in understanding study-specific terminology, and, of course, ethical and regulatory constraints.

The results have practical relevance for clinical trial programmers who are seeking to improve the quality of their code logic and reduce the risk of programming errors during critical stages of clinical reporting. The presented approach augments existing validation practices by focusing on the logic-semantic aspect, which has remained beyond the scope of automated analysis so far.

INTRODUCTION

In clinical trial programming, logical errors which are also called semantic errors occur when a SAS® program runs without crashing but produces incorrect results because the implemented logic differs from the intended analysis. These errors are different from syntax ones (which cause immediate program failure) or data issues (which can't be tracked in the program code). A logic error does not terminate the program. Instead, it produces plausible but wrong output, making it a programmer's "worst nightmare" to detect. For example, if the Statistical Analysis Plan (SAP) specifies an algorithm for deriving a safety endpoint and the SAS code misapplies that rule, the output may look reasonable even as it violates the SAP's objective. Detecting such mistakes is challenging because no SAS error messages will appear.

Current quality control (QC) practices in the pharmaceutical industry have well-known approaches in catching these semantic issues. The gold standard is independent double programming: two programmers implement the same analysis independently based on the SAP, then compare the outputs (often using SAS's PROC COMPARE). This method can confirm consistency between two results and is appropriate for high-impact analyses. However, double programming doesn't absolutely guarantee the correctness of the final data set. If both programmers interpret a specification in the same incorrect way, they will still produce matching but incorrect results. This scenario would never be revealed by PROC COMPARE. In practice, mistakes still occur even with precise double programming, often due to ambiguous specifications or code that remains unexecuted because no data meets the required conditions. In other words, existing QC focuses on detecting differences in outputs rather than verifying alignment with the intended analysis logic. PROC COMPARE itself only highlights mismatches between two datasets and unfortunately it cannot judge whether either dataset is correct with respect to the SAP. In the end, human judgment is required to reconcile code output with the protocol and SAP, which leaves a gap for human error.

To close this gap, we need to understand both the requirements and the code in a more semantic way. This is exactly where AI can help. Recent developments in large language models (LLMs) like ChatGPT® have shown good progress in understanding natural language and programming code. ChatGPT, for instance, can parse textual instructions and generate or analyze code, effectively acting as a bridge between human language and software logic. This capability suggests an opportunity to improve part of the semantic validation: an AI that "reads" the SAP and the SAS program could assess if the program's logic faithfully implements the described analysis.

Early explorations in our field have hinted at this potential, for example, using ChatGPT to review SAS code has been noted to provide helpful insights, often more reliably than having it write code from scratch. By leveraging an LLM's understanding of language and context, the proposal is to augment the QC process with an automated semantic check.

In the following sections, an approach that uses ChatGPT to detect logical errors in clinical SAS programs will be introduced, aiming to ensure that code logic is consistent with the SAP and protocol. This approach offers an additional layer of quality control focusing on what the code is doing (semantically), not just whether two outputs match, thus dealing with a critical gap in current clinical programming validation practices.

INTEGRATION OF CHATGPT IN SAS ENVIRONMENT

To integrate ChatGPT connection directly into SAS Studio, the code provided below will be used. However, the first thing to do is to visit the link provided at platform.openai.com, then create an account using an email address. The next step is to create a secret API key that allows access to the API. Make sure to copy and securely store this API key for future use.
In our SAS program the goal would be to create a macro code below, where only two macro variables should be adjusted: **&API_KEY** and **&QUESTION**. The **&API_KEY** macro variable will hold the Secret API key, while the **&QUESTION** macro variable will capture the user's input, which serves as a question or search query. Users can adjust model version as well, if a new one is released.

```
%macro RunGPTquery(query =);
%let API_KEY= sk-XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX;
%let QUESTION = %str(%"&query.%");

/* Add model to request */
filename in temp;
data _null_;
file in;
put;
put "{";
put  '"model": "gpt-5", "messages": [{"role": "user", "content": '"&question }]";
put "}";
run;
/* Reference the file as in */
filename resp "%sysfunc(getoption(WORK))/echo.json";
proc http
 method="POST"
 url="https://api.openai.com/v1/chat/completions"
 ct="application/json"
 in=in
 out=resp;
 headers "Authorization" = "Bearer &api_key.";
run;
/* Parsing the response and output */
libname response JSON fileref=resp;

proc report data= response.choices_message;
column content;
define content / display "" style(column)=[cellwidth=6in fontsize=10pt asis=ON];
run;
%mend RunGPTquery;
```

The user should ensure that the secret API key is not enclosed in quotation marks. Additionally, retain the percentage (%) signs and quotes within the **&QUESTION** macro variable as they are essential components for its proper functioning.

The next step is to include the macro code from the external file and call it with a query.

```
%include "/home/u43526594/SAS & ChatGPT/RunGPTquery.sas";
%RunGPTquery(query = SAS code to create boxplot by treatment group and save it to .pdf
format);
```

After executing the code, the output will be generated with the response of the query.

```
Here is an illustrative example of how you might generate the boxplot as requested:
```sas
ods graphics on;
ods pdf file="c:/my_folder/my_boxplot.pdf";
proc sgplot data=my_dataset;
   vbox my_variable / category=treatment_group;
   title "Boxplot by Treatment Group";
run;
ods pdf close;
ods graphics off;
```
```

Now we ensure that connection with AI model works directly from our SAS environment, that is why we can move to the implementation of our concept.

METHODOLOGY

This project employs ChatGPT (specifically the newest GPT-5 model) as a semantic validation tool integrated into the SAS 9.4 programming environment. The core idea is to verify that SAS code logic aligns with unstructured documentation such as the Statistical Analysis Plan (SAP) or study protocol. To achieve this, we will use the Integration of ChatGPT into the SAS IDE environment through API as introduced in reference (1) for our further steps since it gives a number of advantages such as real-time assistance, efficiency, and contextual relevance. Briefly, the SAS environment calls the ChatGPT API (OpenAI's® endpoint) programmatically via PROC HTTP, sending a query that includes both the code fragment and the corresponding part from the SAP. Each validation task is handled as a separate prompt to the model: the SAS program prepares a code snippet along with the associated SAP rule text, and these are submitted together in a single API request. The response from ChatGPT, which typically highlights any discrepancies or confirms correctness, is then captured and made available for the programming team to review. In summary, the general workflow involves SAS initiating an API call with the code and SAP text, and ChatGPT returning a natural language report on whether the code correctly implements the specified analysis logic.

PROMT ENGINEERING STRATEGY

Careful prompt design is crucial to focus the model on logic validation. Each prompt to ChatGPT should be structured with clearly defined sections. First, role instruction is given to set the context and tone. For example, we begin with a system prompt like: "You are a senior statistical programmer or an FDA auditor reviewing SAS code for compliance with the SAP." This primes the model to adopt a critical and detail-oriented mindset. Next, the user prompt is organized into two labelled parts: one for the SAP rule and one for the SAS code. We introduce the SAP citation, for instance with a header such as "**SAP Rule:**", followed by the relevant text. Then we introduce the "**SAS Code:**" section, providing the code snippet (we can keep comments inside the code). The prompt concludes with a direct validation instruction to the model, such as: "Identify any discrepancies between the SAP rule and the SAS code logic. Focus on the computation and logic only and ignore coding style or syntax." This final instruction directs ChatGPT's attention to the semantic comparison task. Overall, this prompt engineering strategy ensures the model knows its role (expert reviewer), the context (SAP vs. code), and the task (logic validation), hence minimizing off-topic responses.

CODE INTEGRATION

To illustrate our methodology let's assume we are at the early stage of production code development and don't have much data on our study. However, we still want to ensure that our codes will work correctly when new data is available. Let's check the Treatment-Emergent Adverse Event section.

SAP Rule (example):

```
%let SAP_INSTRUCTIONS = %nrstr(An adverse event (AE) is considered treatment-emergent
(TEAE) if the event start date occurs on or after the date of first study drug
administration (TRTSDT), and starts not later than 30 days after the last dose or
follow-up anti-cancer data, whichever occurs earlier. Events that start before TRTSDT
should not be classified as TEAEs.)
```

SAS Code (with logical issue):

```
%let CODE_PART = %nrstr(
* Get earliest Follow-up anti-cancer date;
proc sql;
  create table fuac_earliest as
  select distinct USUBJID,
         min(CMSTDTC) as FUACDT format=date9.
```

```
  from   sdtm.cm
  where upcase(CMCAT) = 'FOLLOW-UP ANTI-CANCER THERAPY'
  group by USUBJID;
quit;

* Get ae data;
proc sort data=WORK.ADAE out=ae_dates;
by USUBJID;
run;

data adae_teae;
  merge ae_dates(in=inae)
        fuac_earliest;
  by USUBJID;
  if inae;

  /* Get earliest date from follow-up and end+90 days date
  TRTEDT - date of last dose, TRTSDT - date of first dose
  ASTDT - start date of AE*/

  AE_END = min((TRTEDT + 30), FUACDT);

  /* Classify TEAE per SAP definition */
  if TRTSDT <= ASTDT and ASTDT <= AE_END then TEAEFL = 'Y';
  else TEAEFL = 'N';

run;
);
```

Prompt which we are going to send via API to ChatGPT:

```
System message: You are a senior statistical programmer or an FDA auditor validating
SAS code against SAP rules. Focus only on logic and ignore coding style or syntax.
User message:

<<SAP RULE>> &SAP_INSTRUCTIONS.
<<SAS CODE>> &CODE_PART.

<<TASK>>
Does this part of a SAS code have any logical issues with respect to the SAP rule?
Identify any show only direct issues with SAP
```

Below we can see the response:

```
No direct logical issues. The code implements the SAP rule correctly by:
- Defining the TEAE window as TRTSDT ≤ ASTDT ≤ min(TRTEDT+30, FUACDT), inclusive.
- Ensuring events starting before TRTSDT are not classified as TEAEs.

No SAP-defined condition appears to be violated in this logic.
```

As expected, ChatGPT identified no logical issues in this part of the code, which is consistent with the SAP rule. However, it should be noted that when the condition was initially written as TRTSDT <= ASTDT <= AE_END, the GPT-5 model repeatedly flagged this expression as problematic across several runs. This behavior highlights potential challenges in how LLM handles compound logical operators. A more detailed discussion of these limitations is provided in the later sections of the paper.

CASE STUDY: USING GPT-5 MODEL TO DETECT LOGICAL INCONSISTENCY IN BOR DERIVATION

This case study demonstrates an end-to-end example of using OpenAI's GPT-5 language model (via ChatGPT) to identify a logic error in a clinical SAS program. The scenario involves deriving the **Best Overall Response (BOR)** in an oncology trial. By feeding a Statistical Analysis Plan (SAP) part and a SAS code part into GPT-5, we show how the model can detect a semantic discrepancy that standard validation tools would likely miss. The aim is to illustrate GPT-5's role as an automated semantic quality control tool, reasoning over both unstructured SAP text and structured code.

Rules from SAP:

The SAP part below reflects a typical requirement drawn from RECIST 1.1 guidelines:

- BOR is defined as the best tumour response from treatment start until progression; responses are prioritised as **CR > PR > SD > PD**.
- A Complete Response (CR) or Partial Response (PR) must be confirmed by a subsequent tumour assessment of CR or PR, at least four weeks later, with no intervening Progressive Disease (PD).
- Unconfirmed CR/PR should not be counted as confirmed responses. Stable Disease (SD) and PD follow their standard definitions.

SAS Code Example:

Below a simplified SAS program is shown, it attempts to derive BOR for each subject. The logic is deliberately incomplete: it selects the best observed response category (CR, PR, SD, or PD) but fails to implement the confirmation rule from the SAP.

```
proc sort data=responses out=resp_sorted;
    by subject response_date;
run;

data bor_derivation;
    set resp_sorted;
    by subject;
    retain best_resp 'PD';              /* initialise BOR as worst (PD) */
    if first.subject then best_resp = 'PD';
    /* update best_resp based on hierarchy CR > PR > SD > PD */
    if response = 'CR' then best_resp = 'CR';
    else if response = 'PR' and best_resp not in ('CR') then best_resp = 'PR';
    else if response = 'SD' and best_resp not in ('CR','PR') then best_resp = 'SD';
    if last.subject then do;
        bor_resp = best_resp;           /* assign derived BOR result */
        output;
    end;
run;
```

In this program, tumor responses are processed in chronological order for each subject. The best_resp variable retains the highest-priority response observed (CR first, then PR, then SD). At the final record for each subject, bor_resp is assigned to this value and output. The code therefore labels a patient's BOR as CR or PR based on a single observation, without checking whether the response is confirmed. According to the SAP, a CR or PR must be confirmed by a subsequent CR/PR assessment at least 28 days later. The absence of such check constitutes a logical inconsistency with the SAP.

GPT-5 Prompt Construction:

To evaluate the code against the SAP, a prompt for GPT-5 containing the SAP rules and the SAS code was constructed. The prompt included:

- A system message: "You are a clinical data analysis assistant tasked with verifying that a SAS program correctly implements an SAP rule."
- A user message containing the SAP part with rules (summarized above) and particular SAS code for BOR creation, each clearly labelled.

- A task instruction: “Identify any logical inconsistencies between the SAP’s BOR criteria and the SAS code. Explain any discrepancies.”

GPT-5 Response directly to SAS Environment:
GPT-5: “The SAS code and the SAP excerpt are **inconsistent** in how they handle confirmed responses for BOR. According to the SAP, a CR or PR needs confirmation by **a subsequent assessment** (≥ 28 days later, no intervening PD). However, the SAS code does not implement any confirmation check; it simply takes the highest response observed. This could lead to inflated efficacy results. The SAS program should include additional logic to ensure that any CR/PR is backed by a confirmatory assessment; without this, the derived BOR values do not meet the SAP’s criteria.”

We can make a conclusion from the CGPT response that it correctly identified the missing confirmation logic as the core issue. It noted that the code would misclassify unconfirmed responses as confirmed, potentially overstating response rates. Analyzing this small piece of SAS code, the model did not raise irrelevant concerns. It focused specifically on the semantic rule mismatch. This example demonstrates that GPT-5 model is able to understand the meaning of the SAP requirements and the code.

INTEGRATION OF MORE COMPLEX LOGICS

GPT-5 model has correctly identified the absence of a confirmation rule in the SAS code and has accurately explained the potential impact of that omission. The model has focused on the semantic mismatch between the SAP and the code and avoided irrelevant comments. This confirms that GPT-5, when provided with a clear rule and a short piece of code, can act as a semantic reviewer and highlight logic discrepancies.
However, subsequent testing with a more complex scenario revealed important limitations. We supplied GPT-5 with a larger set of RECIST 1.1 rules: where below SAP rules have been implemented by a complex SAS code for more than 300 rows of code and different macro calls.

Table 1.1: RECIST v1.1 Best Overall Response when confirmation of CR and PR required

Overall Response first timepoint	Overall Response subsequent timepoint	Best Overall Response
CR	CR	CR
CR	PR	PR
CR	SD	SD provided minimum criteria for SD duration met, otherwise, PD
CR	PD	SD provided minimum criteria for SD duration met, otherwise, PD
CR	NE	SD provided minimum criteria for SD duration met, otherwise, NE
PR	CR	PR
PR	PR	PR
PR	SD	SD provided minimum criteria for SD duration met, otherwise, NE
PR	PD	SD provided minimum criteria for SD duration met, otherwise, PD
PR	NE	SD provided minimum criteria for SD duration met, otherwise, NE
SD	SD	SD provided minimum criteria for SD duration met, otherwise, NE
SD	PD	SD provided minimum criteria for SD duration met, otherwise, PD
SD	NE	SD provided minimum criteria for SD duration met, otherwise, NE
NE	NE	NE
No post-baseline tumor assessments		Not Done

Note: PR or CR assessments separated by one NE assessment (e.g., PR-NE-PR or CR-NE-CR) will be considered confirmed PR or confirmed CR, respectively.

Covering fourteen confirmation scenarios involving CR, PR, SD, PD, and NE, along with special cases such as PR–NE–PR and CR–NE–CR and a much longer SAS program implementing these rules. The program included macro definitions, multiple PROC SQL steps for look-ahead logic and intermediate datasets for confirmation dates. When we prompted GPT-5 with this comprehensive SAP rules and the corresponding macro-based code, the model did not reliably detect logical mismatches. In several runs, it failed to identify any issues at all and in other cases it flagged portions of the code that were, in fact, correct. It also offered general suggestions that were unrelated to the core logic. This inconsistency suggests that the model’s ability to analyze complex code significantly decreases as the context length and logical depth increase.
The difference between the simple and complex cases illustrates an important point: GPT-5’s semantic understanding appears adequate for short, linear code segments but is challenged by lengthy, modular programs that involve macro resolution, multiple look-ahead checks, and embedded condition chains. As a result, GPT-5 can assist with detecting obvious logic errors in concise examples, its reliability drops sharply in real-world programming tasks that use macros and layered logic. This marks the need for the independent verification and cautious interpretation

when applying large language models to critical clinical code.

LIMITATIONS IN MORE COMPLEX SCENARIOUS

The experiment with the more complex RECIST rules exposed several specific issues:

- Missed logical mismatches - GPT-5 often failed to recognize that the code did not fully implement certain RECIST scenarios, such as implementing minimum SD duration before PD. These omissions went undetected, suggesting that the model could not follow the multi-step logic across multiple data steps and macro calls.

- False positives - In several instances, GPT-5 incorrectly flagged correct logic as not correct. For example, it misinterprets the use of intermediate datasets to derive confirmation dates, even though these steps were essential to implement the SAP rules. These misclassifications indicate that the model may misinterpret legitimate code constructs when the program is long or includes macro codes.

- Limited context memory - The model's responses suggested that it could not "remember" earlier parts of the code when evaluating later segments. As a result, GPT-5 sometimes proposed modifications that contradicted earlier logic. This is a known limitation of current language models: their effective context window is finite, and long code snippets can exceed their reasoning capacity.

These observations coordinate with notes in the literature: ChatGPT and similar models can offer useful insights, but their responses are not error-free, particularly in technical domains like SAS programming. Users must check and validate any AI-generated feedback before implementing it in real work. In the context of complex clinical code, this means maintained human oversight and, where possible, breaking large programs into smaller segments for AI review. Advanced techniques such as rule decomposition and iterative prompting may also be needed to help the model maintain context.

Summing up, our experiments suggest that GPT-5 can be a helpful tool for semantic validation in straightforward cases but is not yet reliable for complex programming logic. The model's tendency to overlook mismatches or produce false alarms in longer, macro-driven code underscores the importance of human expertise in clinical programming quality control.

REGULATORY CONSTRAINTS ON AI/LLM USAGE

The ethical and regulatory constraints for using large language models (LLMs) such as GPT 5 in clinical trial programming are formulated by established data protection rules and emerging AI specific guidance. At the core of clinical research ethics is the responsibility to protect participant confidentiality.

The ICH E6 Good Clinical Practice guideline specifies that "the confidentiality of records that could identify subjects should be protected, respecting the privacy and confidentiality rules in accordance with the applicable regulatory requirements".

The forthcoming E6(R3) revision says that investigators must "protect the privacy and confidentiality of personal information of trial participants in accordance with applicable regulatory requirements on personal data protection" and notes that trial data systems must prevent unauthorized access, disclosure or alteration. These regulations offer computerized systems, which should ensure secure and attributable access, controlled permissions, and mechanisms to prevent accidental loss or destruction. Accordingly, any integration of LLM models (especially external) with SAS code should be designed so that only de-identified specifications and code are sent to the model, and all patient level data remain within validated clinical databases. Internal development of any AI system should be carefully designed as a critical software.

Regulators also emphasize that electronic systems used in clinical development must be validated and auditable. The U.S. FDA's Part 11 guidance notes that electronic records systems require validation, audit trails, record retention and copying. Industry references such as the GAMP guide and ISO standards are cited as best practice frameworks for validating automated systems. These requirements mean that any AI assisted QC process must maintain version control of the model, retain the exact prompts and outputs for auditing, and ensure that the overall system remains in the state of control.

AI specific guidance highlights the need for human oversight and risk-based controls. The World Health Organization's ethical AI principles emphasize protecting human autonomy, ensuring transparency and explainability, and developing accountability. Good Machine Learning Practice (GMLP) principles, developed by the FDA, Health Canada and MHRA, require multidisciplinary oversight, good software engineering and security practices, independence between training and test data, and continuous post deployment monitoring of model performance. For high-risk applications such as clinical decision support, the EU AI Act mandates strict compliance, including data quality, transparency, human oversight and robustness.

Despite improvements, LLMs remain sensitive to hallucinations. OpenAI notes that hallucinations - confident yet false statements - are a fundamental challenge. GPT 5 "has significantly fewer hallucinations … but they still occur" was stated by OpenAI official documentation. Our case study confirms following: GPT 5 reliably identified simple logic errors but struggled with more complex, macro driven RECIST code and occasionally flagged correct logic as incorrect. Thus, AI outputs should be treated as advisory, not definitive. Sponsors must log prompts and responses to

ensure traceability and reproducibility and should re-evaluate models when they are updated, because evolving model weights can alter outputs for identical inputs.

Summing up all the regulatory constraints mentioned above, integration or development AI models such as GPT-5 into clinical programming must respect existing data privacy and electronic record regulations, implement strong validation and audit trails, and follow emerging AI ethics guidelines that focus transparency, accountability and human control. AI tools should augment, not replace, human review: they can accelerate the detection of logic inconsistencies but cannot be the one and only instrument. By masking patient identifiers, maintaining audit trails, versioning models and prompts, and combining AI checks with traditional QC, sponsors can develop their own AI models and utilize the benefits of generative AI while meeting regulatory expectations.

CONCLUSION

In summary, this paper proposes a semantic quality-control workflow that augments existing SAS programming practices with large language models. Traditional QC methods focus on detecting discrepancies between outputs, but they cannot determine whether either output matches the underlying Statistical Analysis Plan (SAP). By sending both the relevant SAP text and the corresponding SAS code to a GPT-5 based service, the approach allows the model to act as an additional reviewer who reads the requirements and the code and reports on the logic. Careful prompt engineering, defining the model's role, separating SAP rules from code and instructing it to focus on logic helps improve the model's responses.

The case study on deriving Best Overall Response illustrates the promise and the limits of this technique. When given a concise SAP rule that CR/PR must be confirmed and a short SAS program that omitted the confirmation check, GPT-5 correctly identified the semantic discrepancy and explained its impact. However, experiments with a 300-line macro-driven program implementing numerous RECIST v1.1 scenarios showed that the model often missed logical mismatches, produced false positives and lost context. These failures reflect the finite context window and reasoning capacity of current language models and underscore that they are best worked on linear code segments.

Furthermore, the paper notes that integrating LLMs into clinical workflows must respect data-privacy regulations, maintain audit trails and ensure human oversight. GPT-5's tendency to hallucinate or misinterpret complex logic means its outputs should be treated as advisory rather than definitive. With these safeguards, ChatGPT or any other well designed LLM model can serve as a valuable instrument for catching obvious logic errors and accelerating code reviews, but it cannot analyze complex logic or replace the experience of clinical programmers. Human expertise, version-controlled processes and iterative validation remain imperative.

I am glad to encourage the audience to actively explore the integration of AI language models like ChatGPT into their development processes and find their own and new ways of its usage.

REFERENCES

1) Skliar Illia. 2023. "Boosting SAS Programming Efficiency with ChatGPT: A Clinical Trials Perspective." PHUSE EU Connect, Birmingham: PHUSE.
URL: https://phuse.s3.eu-central-1.amazonaws.com/Archive/2023/Connect/EU/Birmingham/PAP_CM04.pdf.

2) Skliar Illia. 2024. "Bridging AI and Clinical Research: A New Era of Data Management with ChatGPT" PharmaSUG 2024 Baltimore.
URL: https://pharmasug.org/proceedings/2024/SD/PharmaSUG-2024-SD-200.pdf

3) OpenAI platform documentation
URL: platform.openai.com/docs/quickstart/build-your-application

4) OpenAI platform documentation – "Why language models hallucinate"
URL: https://openai.com/index/why-language-models-hallucinate/

5) A Global Perspective on AI for Life Sciences Part 1: AI Applications, Ethics, Regulations, Guidances, and Business Impact.
URL: https://globalforum.diaglobal.org/issue/august-2025/a-global-perspective-on-ai-for-life-sciences-part-1-ai-applications-ethics-regulations-guidances-and-business-impact/

6) E6(R2) Good Clinical Practice: Integrated Addendum to ICH E6(R1) Guidance for Industry, 2018
URL: fda.gov/media/93884/download

7) CFR – Code of Federal Regulations Title 21, 2023
URL: accessdata.fda.gov/scripts/cdrh/cfdocs/cfcfr/CFRSearch.cfm

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Please contact the author at:

Illia Skliar
Statistical Programmer/Analyst
Intego Group, LLC
8 Manizer street
Kharkiv, Ukraine / 61000
Work Phone: +38 044 500 7020 ext. 2546
Email: Illia.Skliar@intego-group.com