Gemini-Driven Automated Prompt Engineering for Mining High-Volume Sustainability Reports at ING Bank

Author: Vitalii Fishchuk
University of Twente, EEMCS
v.fishchuk@student.utwente.nl

Abstract*—LLMs are increasingly used for mining long, heterogeneous reports, increasing the demand for prompt engineering; yet, scalable manual prompt engineering is costly. This study evaluates automated prompt optimization for information extraction at ING Bank. The author adapts the ProTeGi automated prompt optimization framework to the information extraction use case and introduces Gradient Verification (GradV). This decision gate filters LLM feedback, facilitating faster convergence of LLM-feedback-based prompt optimization methods. The work also introduces a transparent prompt enrichment (PE) framework, which converts verified LLM feedback into modular instructions. Using Gemini Flash 2.0, the author extracts absolute emissions of Scope 1, assurance of Scope 3, and reporting period from 141 anonymized annual sustainability reports. On the test set (n=49), the improvements over the initial prompts were 27 percentage points (pp) for S1, 4pp for S3a, and 10pp for Rp; after the Holm-Bonferroni correction, only S1 remains statistically significant (one-tailed exact binomial test on discordant pairs, with stepwise significance levels of 0.0167, 0.025, 0.05). Optimization also improved stability, reducing run-to-run response variability on some variables under identical conditions. Furthermore, optimized prompts displayed similar accuracies to analyst-designed prompts on some targets, clearly highlighting the potential for reducing manual effort. In general, automated prompt optimization has shown potential to be a feasible and scalable alternative to manual prompt engineering for long-context and long-input information extraction in enterprise scenarios for certain target variables.

I. Introduction

Recent developments in Natural Language Processing (NLP) have led to an increase in the popularity of large language models (LLMs) based on the transformer architecture [1], creating numerous opportunities to automate tasks previously considered human-only.

LLMs have demonstrated remarkable performance on a wide range of tasks, with few-shot learning and prompt engineering showing a clear improvement in performance over basic prompting [2], [3]. In addition, prompt engineering was found to outperform more expensive task-specific fine-tuning in some domains (e.g., medical QA) [4], [5].

However, previous work shows that even small changes in the prompts, such as adding an empty space or specifying different output formatting, can result in a completely different result [6]. Furthermore, small changes in the phrasing of the prompt can even change the underlying reasoning behavior of LLMs [7].

Combined with a high-dimensional prompt space [8], scalable manual prompt engineering is hardly feasible, revealing the need for automated black-box prompt optimization strategies. However, a gap appears in the literature regarding the practical application of automated prompt optimization techniques in complex enterprise scenarios that involve lengthy input tasks and contexts. The aforementioned scenarios further impose requirements on scalability toward a high number of tasks and prompts, emphasizing the demand for cost-effective solutions. One such task is extracting information from lengthy reports, which perfectly illustrates the problem of long context windows and long input tasks (documents, each hundreds of pages long). Due to prevailing manual labeling of the data, the data sets in such cases are extremely limited, undermining the potential applicability of automated prompt optimization and also the ability to conclude the effectiveness of the said methods.

This study presents the above scenario, exploring the feasibility of automated prompt optimization in such tightly constrained practical cases, addressing the problem at the cutting edge of LLM-backed enterprise automatization technology. The information extraction workflow is represented by the problem of extracting a set of predefined variables from long annual sustainability reports, submitted to the ING Bank's Wholesale Banking Advanced Analytics department. The au-

thor experiments with the well-established automated prompt optimization method (ProTeGi [9]); introduces a conceptual framework for straightforward and transparent Prompt Enrichment (PE); and proposes a Gradient Verification (GradV) procedure, complementing existing LLM-feedback-based optimization methods. Performance (accuracy) of auto-optimized prompts on annual sustainability reports is compared to both initial and manually engineered prompts, concluding the applicability of automated prompt optimization procedures in the aforementioned constrained scenarios. The code is available on GitHub [1]. In contrast, the data sets are proprietary and not disclosed.

II. Literature and value of this paper

Studies on automated prompt engineering have skyrocketed in recent years, following the wide adoption of LLMs. This paper categorizes the abundant papers based on the following proposed components: optimization target, operator, search strategy, initialization, and feedback signal.

A. Target of optimization

With the growth in LLM acceptance, the definition of prompt has also evolved. From the prompt defined as a single query to the status quo prompt templates consisting of multiple components. A recent study [10] summarizes the existing prompt templates in the following list of prompt components:

- *Profile*: Persona/Role assigned to the model in the prompt.
- *Task instructions*: Task description, including system-level instructions.
- *Workflow*: Reasoning steps and instructions (if any).
- *Context*: Additional useful information, domain-level knowledge, insights, and other similar components.
- *Examples*: Few-shot examples added to the prompt to illustrate the task.
- *Output format/style*: Instructions covering expected format and/or style of the output.
- *Constraints*: Any constraints the model must follow when generating a response.
- *Other*: Any other components, including recaps, experiments, or skill suggestions.

The literature focusing on profile optimization is scarce, but some studies show that LLM-generated personas can improve performance on certain tasks [11] or in multi-agent setups [12]. Another study notes that adding personas to system prompts specifically does not guarantee performance improvement; proper selection is challenging, and the impact of personas is highly variable [13]. Personas can also surface the underlying deep-running biases present in LLMs, hindering performance on reasoning tasks [14]. Simultaneously, LLM-generated personas are shown to display higher stability than those crafted manually [15], highlighting the potential for automated profile optimization.

In contrast to profile optimization, the literature on task instruction optimization is abundant, including, but not limited to, the following methods:

- *APE* (Automatic Prompt Engineer), where an instruction search is performed by proposing new instructions and scoring them on the data [16].
- *ProTeGi/APO* (Prompt optimization with textual gradients/Automatic Prompt Optimization), where the LLM iteratively rewrites multiple prompts, using the so-called "textual gradients" - LLM critiques of the said prompts [9].
- *TEMPERA* (Test-time prompting via Reinforcement Learning), where the authors train a test-time prompt editing function, which then rewrites the prompt to better suit the task [17].
- *OPRO* (Optimization by PROmpting), where LLM proposes new prompt candidates, based on previous prompts and their respective scores [18].
- *GrIPS* (Gradient-free Instructional Prompt Search), where instead of rewriting the entire prompt, phrase-level edits are applied randomly and iteratively, searching for the best modifications through greedy search strategies [19].
- *AutoHint* (Automatic Prompt Optimization with Hint Generation), where the prompt is enriched with additional task instructions, derived from LLM feedback formed on incorrect examples [20]
- *AMPO* (Automatic Multi-Branched Prompt Optimization), where the prompt is enriched with conditional statements derived from multi-branched summarization of error-patterns observed on failed data points [21].
- *PROMST* (PRompt Optimization in Multi-Step Tasks), where the prompt is optimized based on LLM feedback and scoring, obtained by following human-designed feedback rules [22].
- *IBPC* (Intent-based prompt calibration) where the prompt is optimized iteratively based on synthet-

[1] https://github.com/Lolya-cloud/Ape-ING-2025

ically generated examples that are labeled by a human or LLM during the runtime [23].

Workflow optimization is less common, although methods such as AutoHint [20] and PROMST [22] can produce multi-step instructions, especially the latter. Other methods, such as Tree of Thoughts [24], Self-Consistency [25], and Automatic Chain of Thought [26], utilize LLMs to generate a chain of instructions (reasoning) and select the best-performing paths.

Example optimization is fairly common, with notable algorithms being: [27], where the authors train a reward-based example retriever, which evaluates the quality of the example and adds the best ones to the prompt; Iterative Demonstration Selection [28], where the most suitable examples are selected based on similarity to prior LLM-generated reasoning paths; PromptWizard [29], where, in contrast to the previous two methods, examples are subjected to optimization rather than selection procedure. PromptWizard employs a mutate-critique-synthesize approach, powered by LLM agents, to iteratively improve few-shot prompts (both instructions and examples, including reasoning), resulting in synthetically generated examples that best illustrate the task.

The output format component is rarely considered in the optimization literature, yet it can significantly affect the result, even for larger LLMs. In particular, formatting in XML/CSV/YAML can reduce accuracy in some settings; the effect is present even in large models, while the said models can be robust to other small semantical differences and prompt variations [6].

A notable distinction in prompt-type importance must be made for genetic algorithms, exemplified by EvoPrompt [30], PromptBreeder [31], and PhaseEvo [32]. Genetic algorithms begin with a population of prompts, rather than a single seed, and hence directly benefit from prompt diversity. Naturally, initializing the starting population with a diverse collection of prompt types allows for simultaneous inclusion of multiple prompt components in the optimization process.

Furthermore, since genetic prompt optimization focuses on prompt features (parts that differ between parents) rather than the entire prompt, the final optimized prompt is likely to be more complex and contain multiple aforementioned components as a side effect of optimization. In other words, genetic algorithms perform a true exploration of the parameter space, rather than exploitation of a particular part. Consequently, the methods often show higher accuracy with increased evaluation costs [30]–[32]. PhaseEvo [32] attempts to address the cost problem by adding exploitation stages between evolution steps using LLM-guided edits, conceptually similar to the textual gradient technique proposed in ProTeGi/APO [9]. This directly facilitates faster convergence, which, in turn, substantially decreases the costs.

Although most of the papers have a clear focus on one or two prompt components (usually task instructions, sometimes examples), most methods can be adapted to be prompt-type agnostic. Overall, a strong focus on the task-instruction component and some interest in the example component are observed in the literature.

B. The operator

The optimization operator refers to the method that modifies the units of optimization, usually prompts (or prompt components). The literature presents multiple:

- *LLM rewrites and appends:* Arguably one of the most popular methods, where an LLM plays the role of the operator by rewriting/modifying/appending to the prompt. This allows simplifying prompt interaction and stepping away from traditional token-based and rule-based text edits, automatically preserving prompt coherence and interpretability. Some of the methods that apply this operator are: ProTeGi/APO [9], APE [16]; OPRO [18], AutoHint [20], IBPC [23], AMPO [21], and many others.
- *Edit rules over token/phrases:* A more traditional NLP approach, where edits are done following a set of defined edit actions (delete, swap, paraphrase, add), performed on a phrase level using a constituency parser (CRF-based) to obtain disjoint phrase-level constituents. The defined operations are then applied to the said constituents. The technique is introduced and applied in the GrIPS algorithm [19]. A similar approach is applied in TEMPERA [17], where the authors use a similar set of operations to modify the prompts at the phrase level.
- *Population-based LLM-mutations:* Similar to LLM rewrites and appends, LLM performs the role of the operator, this time mutating and crossing over the prompts. Notable papers include EvoPrompt [30], PromptBreeder [31], PhaseEvo [32].
- *Learned policies for prompt modification*: RL-Prompt trains a small policy network to generate and modify prompts using black-box LLM rewards [33], often resulting in working but gibberish prompts.

In general, delegating the operand function to LLM agents appears to be one of the most common and straightforward approaches. This enables the generation of meaningful, coherent, and interpretable prompts while simultaneously facilitating effective optimization.

C. Search strategy

A distinction can also be made between existing prompt optimization methods based on the employed search strategy. The search strategy refers to the optimization algorithm itself, including the conceptual difference between exploration and exploitation. These include:

- *Local improvement of a single prompt:* This strategy involves iterative improvement of one or multiple prompts, showing an example of local exploitation (improving the existing starting prompts). Downstream, various methods can be employed, including gradient descent and beam search. Notable methods with this search strategy include ProTeGi/APO [9], GrIPS [19], AutoHint [20], IBPC [23], AMPO [21], and many others.
- *Population-based exploration:* Genetic exploratory techniques that evolve a population of prompts while preserving and accumulating promising features (prompt components) in the said population. Examples are EvoPrompt [30], PromptBreeder [31], PhaseEvo [32]. The last method also employs local improvement in-between evolution stages.
- *Reinforcement learning:* Learning a specific policy that can be applied to optimize the prompts. Notable examples include RLPrompt [33] and TEMPERA [17], where the former learns an edit policy and the latter - a function that modifies the prompt during the test time.
- *LLM as optimizers*. This search strategy refers to directly utilizing LLMs as optimizers by iteratively feeding in prior prompts and solutions, while querying for new optimized prompts. A notable example is OPRO [18].

D. Initialization

Another divide present in the literature is the starting point of optimization. Methods like ProTeGi [9], AutoHint [20], GrIPS [19], and GPS [34] begin with a human-seeded prompt (manually written), often a simple task description. Several methods, while assuming an initial human prompt, note potential sensitivity to the quality of the said prompt; e.g., ProTeGi [9].

In contrast, methods such as APE [16] generate the initial prompt using LLM induction (inducing the prompt from input-output task pairs). Again, a distinction needs to be made specifically for genetic algorithms [30]–[32], as starting with a prompt population in place of a single entity allows for the utilization of a combination of manual and LLM-generated prompts.

Although most of the above methods could be seeded with LLM-induced starting prompts, the impact on the optimization process and result quality is unclear. Furthermore, this strategy would strengthen the dependence on the quality and quantity of ground truth data.

Besides the starting prompt seed, initialization also includes defining a set of meta-prompts — prompts that describe how the LLM should perform any tasks delegated to it during optimization. This is particularly relevant for LLM-based prompt edits and feedback. Meta-prompts are often designed manually by authors, but could be made task-agnostic (see, for example, ProTeGi/APO [9]). Algorithms such as PromptBreeder [31] go a step further and include meta-prompts (mutation instructions in this case) into the optimization process itself, combining them with task-prompts to form a single unit of evolution [31]. However, the optimization and importance of meta-prompts for prompt optimization procedures seem to often be omitted in the literature.

E. Feedback signal

The feedback signal refers to the ”reason for” or ”direction of” prompt change made by the operator. Popular methods include:

- *Task-native metrics*. In this category, task-specific metrics directly serve as the feedback provider. For example, if a prompt has lower accuracy, it should be changed. This approach is used in OPRO [18], ProTeGi/APO [9], GrIPS [19], RLPrompt [33], and many others.
- *LLMs as optimizers or feedback providers*. This refers to another popular method for determining the feedback signal, which sometimes also includes the direction of the change. Under the framework, LLMs are used directly to produce feedback or optimize the prompt. This concept coincides with the ”textual gradients”, defined in [9] as prompt critique/feedback by the LLM agent. Notable algorithms taking this approach are ProTeGi/APO [9], OPRO [18], AutoHint [20], AMPO [21], PhaseEvo [32], and others.
- *Human-alignment*. A more exotic method, where the prompts align with the intentions entered by humans [35].

As is clear from [9] and [19], among other similar methods, native task metrics are not exclusive to LLM feedback signals and are often deployed sequentially. For example, in ProTeGi [9] and AutoHint [20], the task-specific metric is used first to determine the prompt quality; only if the quality is deemed insufficient, the LLM-based feedback signal is executed.

F. Critique of LLMs as feedback providers and evaluators

LLMs can act as both evaluators and feedback providers in different scenarios [36]–[39], especially when applied iteratively, as demonstrated in the Self-Refine method [40]. However, inconsistency remains present, even when conducting evaluations and providing feedback under the same conditions [37]. In addition, the accuracy of the feedback can be questionable due to the widespread hallucination and misinformation problems present in LLM [41]–[44].

One way to tackle the issue is to sample multiple solutions and aggregate them [25]; alternatively, a tournament selection can be performed with pairwise comparisons of candidates [45]. In code generation specifically, sampling of many candidates can improve the results [46].

A similar concept of repeatedly generating multiple feedback pieces is present in the automated prompt optimization literature. For example [9], where multiple feedback pieces are generated, followed by multiple candidate prompts. Although this enables LLM feedback to be an effective component in automated prompt optimization, more complex scenarios might require a significantly higher number of generated samples, leading to a steep growth in optimization costs. Although LLM-ensemble methods could mitigate this (see [25], [45]), they are considered far beyond the scope of the study.

G. Black vs White box

Most of the existing methods discussed in this paper assume a black-box LLM API scenario and optimize the prompt accordingly. In such cases, the query budget and batch size play a high importance. In contrast, methods such as TEMPERA [17] and RLPrompt [33] train a local auxiliary policy network (or a function) to work with the prompt and are less dependent on the black-box assumption. Still, they are fully compatible with black-box LLM models accessible through APIs.

H. DSPy

A notable framework gaining popularity is DSPy [47], [48], a tool for building self-improving LLM pipelines that utilize prompting, reasoning, and fine-tuning. The framework offers a range of methods for optimizing both task-instruction and example-prompt components [49]. In contrast to most of the literature discussed above, DSPy focuses on providing a production-ready LLM pipelining framework, including streamlined prompt optimization, which makes it a pragmatic method for practical enterprise applications.

I. Data dependency

Although the importance of the dataset size varies, many of the surveyed methods depend on or benefit from a labeled dataset. Prompt optimization without assigned ground truth labels might be possible using weak metrics and LLM as a judge, but this lies beyond the scope of the study.

J. Value of this study

A recent survey [50] proposes a complex taxonomy to categorize the most automated prompt optimization methods based on their algorithmic composition. Additionally, the literature review in this paper presents a range of existing automated prompt optimization methods. However, the study identified a significant gap in the literature regarding the practical adoption of automated prompt optimization techniques for complex, real-world enterprise applications. These applications have a fundamentally different goal from those in the field of scientific inquiry, focusing on domain-specific and task-specific optimization rather than the task-general optimization prevalent in the literature.

Furthermore, a literature gap appears to exist regarding the cost-effectiveness and performance of automated prompt optimization compared to manual prompt engineering in complex, real-life scenarios, where inputs can be hundreds of pages long, rather than the small context paragraphs typically used in the literature. Additionally, in such use cases, the ground truth is generally labeled manually by analysts, leading to datasets with limited sizes and potential inconsistencies/errors.

In practice, the dataset characteristics mentioned above can undermine both the prompt optimization process and the conclusiveness of the results, raising doubts regarding the applicability of automated prompt optimization for this and similar use cases. This study aims to address the gaps described above by conducting a case study of automated prompt optimization for extracting information from sustainability reports in the enterprise scenario of ING Bank.

III. Methodology

A. Research Questions and Hypotheses

1) Primary Research Question: How can automated prompt optimization benefit information extraction workflows at ING Bank?

2) Research Questions:

- SRQ1: How effective is prompt optimization based on iterative improvement through the use of LLM-agents on information extraction from sustainability reports?

- SRQ2: How effective is automated prompt optimization compared to expert-managed manual prompt engineering?

3) Hypothesis Development:

- Hypothesis I: LLM feedback-guided automated prompt optimization significantly improves the accuracy of the information extraction workflow compared to baseline prompts.
- Hypothesis II: The accuracy of auto-optimized prompts is significantly different from that of manually engineered prompts.

The study addresses the aforementioned research questions based on a pre-selected primary version of the automated prompt optimization method (ProTeGi), set at ten iterations (a realistic upper bound established beforehand based on expected training time and costs), thereby demonstrating the feasibility of automated prompt optimization for this use case. Other methods and configurations are experimented with and reported, but do not directly contribute to the above research questions.

B. Research Design Overview

The research will evaluate whether automated prompt optimization improves the accuracy of the data extraction from sustainability reports compared to the initial prompts. Furthermore, the performance of auto-optimized prompts will be compared with that of manually expertly engineered prompts, concluding the feasibility of automated prompt optimization in this use case. Finally, the results will be extrapolated to similar real-world applications to determine the feasibility and specificity of automated prompt optimization.

C. Data and Corpus

The study uses 141 sustainability reports submitted to the Advanced Analytics department of ING Wholesale Banking. Due to the sensitive nature of the data, the documents underwent an anonymization process before being submitted to an LLM. This procedure was performed by ING employees prior to the study and is not discussed in the article. Furthermore, since the said preprocessing was performed manually, the number of documents available for this study was limited, resulting in only 141 files.

The reports are proprietary and were anonymized by ING prior to this study; they are not shared. The author releases the code and meta-prompts (Appendix) to facilitate reproducibility on non-confidential corpora.

The set is further subdivided into 64 training, 28 validation, and 49 testing subsets, maximizing the size of the test data set while simultaneously leaving a proportionally decent number of documents for training. Additionally, the small validation data set was reserved for implementation testing and experimentation. The discordance between the reported subset sizes and traditional split proportions stems from the fact that the test data set was later extended when newly anonymized reports became available, prior to any test-time evaluations. Anonymized reports are drawn from the same distribution of submitted reports, justifying the said extension. No prompts were tuned on the test set before or after the extension, preventing any concerns about data leakage.

Each processed document has a corresponding ground truth entry for each of the target variables, manually filled by ING analysts, enabling effective LLM-feedback-based prompt optimization.

The technical characteristics of the data set can be found in Table I. The statistics presented confirm the specificity of the case, characterized by long context windows and lengthy inputs, with a mean file size of around 10-12 MB and an average document length of 140-160 pages.

Set	#D	μ_{size} [MB]	μ_{pages}	μ_{tokens}
Tr	64	12.02 ± 8.96	155.95 ± 108.21	40236 ± 27918
Val	28	13.02 ± 9.12	166.71 ± 149.29	43012 ± 38518
T	49	9.53 ± 6.30	140.49 ± 111.10	36267 ± 28638

TABLE I: Mean statistics with standard deviations and number of documents across datasets. Tokens refer to the number of input tokens per document submitted to the LLM (in this case, Gemini). Gemini internally processes each page of the document as 258 tokens [51].

D. Use-case specificity

The study is performed on the use case of data extraction from large sustainability reports (a few hundred pages long) using Gemini, an LLM known for its long context window. Due to the long context and size of documents, processing time plays a crucial role, constraining the size of the prompt space explored by optimization algorithms. Furthermore, the study observed both the low accuracy of the initial prompt and the Gemini model's inability to extract certain data, regardless of the prompt. For example, extracting certain information present in the tables proved particularly challenging for the model.

In addition, manual labeling of the dataset could potentially introduce errors, highlighting the need for a robust optimization process that is insensitive to outliers and overfitting. The author did not conduct any validation of the dataset/ground-truths, which could have introduced bias. The dataset is taken as-is, mimicking a realistic extraction scenario with manually labeled data.

Additionally, the underlying LLM (Gemini) has exhibited inconsistencies in persistence, where, under the same

conditions, the model returns different results despite being run at a minimized temperature.

The source of the ground truth is another constraint worth noting. Due to the complexity of the topic, the available ground truth labels are produced manually by ING analysts, often requiring several hours of analysis per document, as they involve extracting a large number of variables (and considering the report length). Consequently, the size of the data set is not only a limitation for the study but also for the timely and effective implementation of automated prompt optimization in similar use cases. This study investigates whether automated prompt optimization is feasible with such a limited amount of training data and whether the effects of this improvement are visible on similarly constrained test datasets.

E. Automated Prompt Engineering Framework

Based on the literature and objectives, this paper distinguishes exploratory and exploitative algorithms, recognizing that genetic algorithms are not feasible for the use case due to higher costs and longer training times. In contrast, methods such as ProTeGi [9] also aim to promote global exploration by optimizing multiple prompts simultaneously, while being more cost-effective than random exploration seen in genetics. The simultaneous exploration of multiple prompts could further mitigate the observed inconsistencies in the ground truth, while also addressing LLM persistence problems.

Following these considerations, the study uses a custom implementation of the ProTeGi algorithm [9], adapted from classification to the information extraction use case, as the main iterative prompt optimizer. This fits the scenario requirements of minimal human involvement, zero-shot prompt type, and attempts to mimic aspects of global exploration with minimal cost impact.

1) ProTeGi: An overview of the standard ProTeGi algorithm is presented in Figure 1, where p refers to the prompt and g refers to the ”textual gradient” (a text description of how the prompt should be changed, mimicking a gradient update in the direction of minimization of the loss function). The optimization iteration begins with each prompt in the beam (a set of prompts) being expanded into multiple new prompt candidates, using feedback pieces generated by the LLM from incorrect examples (see Figure 2).

After all of the prompts in the beam are expanded, the produced candidates are aggregated with the original beam into a new set, which is subjected to efficient evaluation techniques aimed at identifying the best prompts while minimizing data usage. This paper utilizes the method used in the original work, the Upper Confidence Bound (UCB1) bandit evaluation [9], [52], a widely applied method for balancing exploitation and

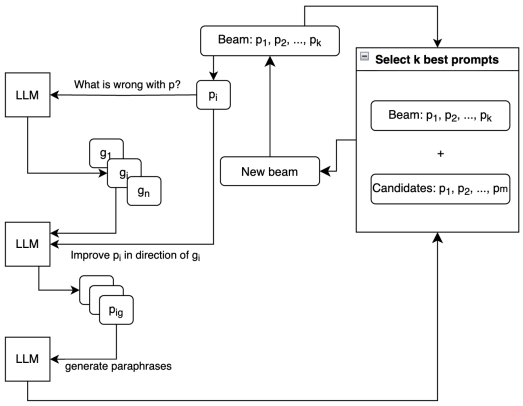

Fig. 1: High-level overview of the ProTeGi algorithm. P refers to prompt, g - to gradient (”textual description of error”). Selection is performed using the UCB-bandits algorithm. In this study, paraphrasing was disabled; the diagram illustrates the general ProTeGi flow.

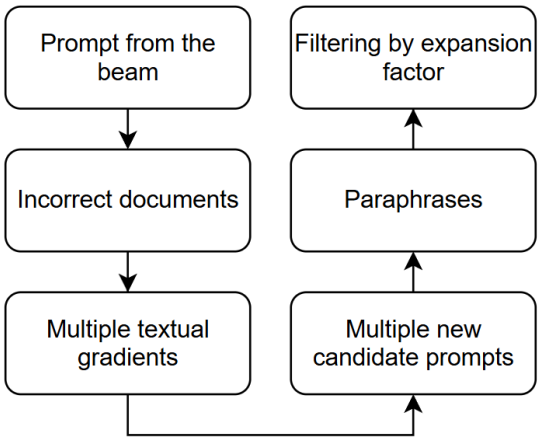

Fig. 2: ProTeGi algorithm for expanding every prompt in the beam. The expansion factor refers to a parameter set beforehand as a threshold for the maximum number of acceptable prompt candidates. The said factor is applied as a filter, randomly trimming the list of candidates. In this study, paraphrasing was disabled; the diagram illustrates the general ProTeGi flow.

exploration in multi-armed bandits. In each evaluation round t, the UCB algorithm selects a prompt candidate p that maximizes

$$\hat{\mu}_p + c\sqrt{\frac{\ln t}{n_p}},$$

where $\hat{\mu}_p$ is the mean score of the candidate prompt p, and n_p is the number of times the candidate has been evaluated. As a result, the evaluation algorithm balances

exploring candidates with high estimated performance and high uncertainty, leading to effective selection of prompts at reduced evaluation cost. Although the original UCB algorithm performs regret minimization rather than best arm identification [9], [53], the ProTeGi paper shows that UCB still outperforms and/or matches other efficient evaluation methods on the prompt selection task [9]. This study employs ProTeGi, utilizing the UCB evaluation version, for efficiency, as the ProTeGi paper [9] has found it to be competitive for selection.

The resulting best prompts replace the beam, and the algorithm proceeds to the next iteration. Finally, after the last optimization iteration, the resulting beam is evaluated on the entire training dataset to select the best-performing prompt.

2) ProTeGi modifications: The original ProTeGi algorithm has shown a stable improvement in performance over the baseline in binary classification tasks [9]. However, this paper deals with the case of information extraction from large documents, displaying a set of differences from the original use case:

- *Information extraction instead of binary classification*. The task performed is information extraction, which complicates the evaluation components of the algorithm and requires custom metric functions.
- *Document size*. The documents submitted range from a few pages to up to five hundred pages in length, and from a few hundred kilobytes to 48 megabytes in size. See Table I for more details.
- *Full context window*. Due to the lack of standardization in sustainability reporting, the required information can be presented in any format, located anywhere in the document, and repeated multiple times.
- *Contradictions in the reports*. In rare cases, the same variable present multiple times in the report can have slightly different or straight-up contradicting reported values (from rounding rules or different context-dependent definitions).
- *Definitions*. Furthermore, due to a lack of standardization, some reports were observed to interpret the definitions of the target variables differently, resulting in inconsistencies in interpretation between different reports.
- *Ground Truth quality*. The ground truth is produced manually, which could potentially introduce errors. Due to the nature of the data, validating the ground truth is complicated, and attempts at prior filtering using LLMs have failed.

Following the above requirements, a custom implementation of ProTeGi is used, introducing the following modifications:

- *Adapting to text extraction*. The algorithm is modified to enable information extraction tasks while maintaining binary document classification (correct/incorrect) by utilizing custom metric functions and custom LLM output schemas. The default metric function is defined as the exact match between the extracted data and the ground truth.
- *Modifying meta-prompts*. The set of meta-prompts was revised to suit the information extraction use case better and facilitate schema usage.
- *Constraining new gradients to a single incorrect example*. Due to the large context window and document size, generating a set of gradients based on multiple documents increased the unpredictability of the underlying LLM model. This led to cases where the LLM did not follow the meta-prompt instructions, halting the optimization process. Hence, the gradients are generated based on a single document each time. Although this constraint enables the model to focus more on individual papers and potentially discover more complex relationships, it can also lead to overfitting to outliers. To mitigate the effect, the full-scale algorithm is run for a larger number of iterations (10), simultaneously improving multiple prompts (4).
- *Limiting new candidate prompts to one per gradient and removing paraphrases*. This study observed exponential growth in costs when chaining parameter values for the number of new candidate prompts and paraphrases. Sequentially, to decrease costs and processing times, it was assumed that optimizing for gradients is more important than performing local exploitation of existing prompts. This allowed the algorithm to prioritize the search for good gradients, addressing the complexity of the case.

  The second option was to implement the maximum expansion factor utilized in the original ProTeGi; however, this could substantially increase the randomness of the optimization process, negatively impacting the reproducibility of the results in the constrained parameter scenario.

Effectively, this study employs prompt optimization, built primarily on the LLM feedback component, to maximize exploration while still utilizing iterative beam search and UCB-bandit evaluation. Furthermore, ProTeGi is customized by increasing the number of iterations to compensate for the complexity of the extraction workflow and observed persistence variability in the

underlying LLM model. The details of the algorithm parameters, generated prompts, meta-prompts, and a demo example of the internal workings of the customized ProTeGi algorithm are provided in the Appendix.

F. Implementation details

This study was conducted from April to September 2025, utilizing Gemini Flash 2.0 snapshot 001 for optimization and inference workflows, due to its popularity in the enterprise scenario and long context capabilities.

The study consciously minimizes stochasticity during information extraction by using near-greedy decoding (temperature 0.01), which is recommended for deterministic tasks [54]. Setting a temperature higher than 0 by 0.01, while nearly empirically indistinguishable, allows the study to avoid potential edge cases of zero division or tie-breaking. The official documentation notes small residual nondeterminism even at zero temperature [55], [56], and hence a choice of 0.01 as stable near zero is justified. In Gemini specifically, the official documentation states that temperature sampling is applied after top k and top p filtering [54], allowing the study to treat temperature as the primary entropy control and keep other parameters set to vendor defaults (see the Appendix for exact values).

In contrast, gradient generation and application (the training component of prompt optimization) were configured with a temperature of 1.0, encouraging diversity and widening the search space. In this part of the pipeline, the top-p was also set to 1.0 (effectively disabling filtering). As this stage is sequential and independent of the extraction step, the change does not affect the validity of the results while further emphasizing the diversity of gradient exploration.

Gemini API also allows one to set a specific seed, in an attempt to increase the likelihood of identical responses for repeated requests [55]. As this is marked as a preview feature and does not yet guarantee a deterministic response [55], the study did not explore the effect of the parameter.

G. Target Variables

Multiple target variables are available per document; this study uses absolute emissions of scope 1, assurance of scope 3, and reporting period (see Table II). Scope 1 absolute emissions refer to the emissions directly produced by the operations of the company submitting the report. Scope 3 assurance refers to the existence of an independent confirmation (audit) for the company's scope 3 emissions (emissions that are not produced directly from the company's operations). The reporting period refers to the date range covered by the submitted report.

All three can be None if the information is not present in the report. Example ground truth entries are shown in Table III. To prevent a zero-float-value interpretation problem for Scope 1 absolute emissions (as zero can be interpreted as "not reported" or "not found"), both outputs of zero and "None" from the model (under the schema) are treated as "not stated or not mentioned". This is justified as none of the documents specifically reports scope 1 absolute emissions as zero (S1 was found to never be zero if reported).

The task descriptions per variable are (used as initial prompts):

- *Scope 1 absolute emissions (S1)*: What is the scope 1 greenhouse gas emission of this company in 2023?
- *Scope 3 assurance (S3a)*: Does the company have scope 3 assurance for the reporting year 2023?
- *Reporting period (Rp)*: Extract the start and end reporting date from the following file.

The output format is constrained using Pydantic data schemas, which enforce the format (see Table II). The schemas are consistent and reused across all experiments and evaluations in this study, thereby addressing potential confounding factors. The detailed programmatic description of the schemas is available in the Appendix.

The diversity of formats shown in Table II enhances the generalizability of the results to different target variables and formats, demonstrating the applicability of the automated prompt optimization process across various information extraction workflows.

Target Variable	**Data Format**
Scope 1 Absolute Emissions (S1)	Float/None: CO_2e Metric Tons
Scope 3 Assurance (S3a)	Binary/None: Yes/No
Reporting Period (Rp)	Start - Str/None: YYYY-MM End - Str/None: YYYY-MM

TABLE II: Target Variables and Their Corresponding Data Formats. Str denotes String

Question	**Example Ground Truth Entry**
What is the scope 1 greenhouse gas emission of this company in 2023?	12345.67
Does the company have scope 3 assurance for the reporting year 2023?	Yes
Extract the start and end reporting date from the following file	2023-01 to 2023-12

TABLE III: Example Ground Truth Entries per Task question

In addition to the initial prompts, the study also compares the prompts generated by automatic optimization with those manually engineered by the analysts. These

prompts represent a status quo solution used in the information extraction use case and showcase realistic accuracy improvement that can be obtained from prompts manually designed by experts.

H. Evaluation Strategy

1) Benchmark Tasks and Baseline Systems: Prompt optimization is evaluated for multiple information extraction tasks, all of which extract one of the three target variables from the submitted report (see Table II). The performance of the auto-optimized prompts is evaluated against the baseline starting prompt (obtained by rephrasing the task description) and manually engineered prompts, designed by analysts. Each of the target variables is evaluated by exact match (under strict unit normalization for S1), aligning with the use-case; however, this can underestimate partial progress in prompt optimization (e.g., partially correct answers for Rp, or answers within a threshold of S1).

2) Quantitative Performance Metrics: The study captures performance changes across prompts as a proportion of documents with correctly extracted information (or accuracy) across the selected datasets. "Correctly extracted information" is specifically defined as an exact match between extracted and expected information (even for the S1 float values), following the use case of information extraction. Secondly, an auxiliary metric is reported, attempting to capture model-induced variations under the same conditions: variance in proportion of correctly classified documents and variance of evaluations (classifications) per document, both across 10 inference cycles. The author further explores persistence by reporting the percentage of documents with a change in evaluation and model response across 10 identical iterations.

3) Cost and Latency Analysis: The cost of the optimization procedure is presented in terms of the total tokens used and the latency, which is the total time spent waiting for the model response. Furthermore, the cost is divided into two categories: inference, which accounts for the tokens spent on performing the parent task (primarily for evaluations); and feedback, which represents the tokens spent on training and improving the prompt.

I. Confidence Intervals

Due to the small sample size and skewed observations (proportions can be close to 1), the study reports the 95% confidence intervals for the proportions of documents calculated using the Wilson score method [57]. The asymmetry of the said method facilitates robustness to small samples and skewed observations [58], which are present in the study. The formula is as follows.

$$\hat{p} = \frac{x}{n}$$

$$CI = \frac{\hat{p} + \frac{z^2}{2n} \pm z\sqrt{\frac{\hat{p}(1-\hat{p})}{n} + \frac{z^2}{4n^2}}}{1 + \frac{z^2}{n}}$$

where: x = number of successes,
n = number of trials (documents),
$\hat{p}$ = $\frac{x}{n}$ (sample proportion),
z = standard normal distribution z-score

J. Hypothesis Testing Procedure

The study conducts hypothesis testing procedures for the two hypotheses using a full-scale long ProTeGi run (10 iterations; see the Appendix for detailed parameters). This version of the algorithm is considered a good representative of the potential performance of automated prompt optimization. It serves as an indicator of the feasibility of prompt optimization compared to initial and manual prompts. Later, the study also reports other variations of ProTeGi and introduces a new method; however, the hypothesis testing procedure is only conducted on the most complete and most expensive optimization algorithm tested, evaluating the feasibility of automated prompt optimization as a whole (represented by the method), rather than a particular implementation.

1) Experimental Conditions and Controls: The prompt is defined as an independent variable, with the associated proportion of correctly predicted documents per target considered a dependent variable. Other variables controlled in the experiments are the LLM model parameters, data points, target variables, and output format. This study also acknowledges the potential existence of confounding variables, such as the persistence and stability of LLM models and the internal black-box processing of output schemas.

2) Statistical Tests and Significance Levels for prompt performance: The tests are performed on paired samples, where different treatments are applied to the same document. Consequently, one- and two-tailed binomial tests are applied to discordant pairs of observations; this is the exact version of McNemar's test for matched binary outcomes, in which only the discordant cells of the 2x2 contingency table (see Table IV) impact the test result [59], [60].

Define **Prompt B** as the prompt to which treatment was applied and **Prompt A** as the corresponding baseline. Let n_{01} be the number of documents on which **Prompt B** is correct and **Prompt A** is not, and n_{10} the number on which **A** is correct and **B** is not (see Table IV). In addition, define the discordant sample size as $N = n_{01} + n_{10}$ and the probability of success.

$$p = P(\text{B correct} \mid \text{discordant}) = \frac{n_{01}}{N}.$$

a) Hypothesis I: LLM feedback-guided automated prompt optimization significantly improves the accuracy of the information extraction workflow compared to baseline prompts.

Define **Prompt A** as the starting prompt and **Prompt B** as **Prompt A** run through the prompt optimization workflow. We test if **Prompt B** is statistically significantly better than **Prompt A**. The test

$$\text{One-tailed:} \quad H_0 : p \leq 0.5 \qquad \text{vs} \quad H_1 : p > 0.5.$$

The significance level is set to a common value of 0.05.

	Prompt B	
Prompt A	1	0
1	n_{11}	n_{10}
0	n_{01}	n_{00}

TABLE IV: Paired 2×2 contingency table used in binomial test. Only the discordant counts n_{01} and n_{10} contribute to the test statistic.

b) Hypothesis II: The accuracy of auto-optimized prompts is significantly different from that of manually engineered prompts. Define **Prompt A** as the manually engineered prompt, **Prompt B** as auto-optimized. We test if **Prompt B** is statistically significantly different than **Prompt A**.

$$\text{Two-tailed:} \quad H_0 : p = 0.5 \qquad \text{vs} \quad H_1 : p \neq 0.5,$$

The significance level is set to a common value of 0.05. Based on the results, a further test on equivalence or non-inferiority might be conducted.

c) FWER correction: The three target variables are analyzed separately for each of the two hypotheses, effectively splitting each of the primary hypotheses into three distinct sets of null and alternative hypotheses. Therefore, the test procedure becomes a multitest, potentially inflating the family-wise error rate [61]. In other words, the probability of making at least one Type I error in the said group increases. To address that, the study applies a standard simple Holm-Bonferroni correction to the significance levels used in hypothesis testing [62]. This allows for strengthening the conclusions of the hypothesis testing procedures without overcomplicating the statistical part of the study. The author conducts exactly two comparisons on the test data set (one per hypothesis), each on three distinct variables. Hence, the Holm-Bonferroni correction procedure is applied with m = 3 for both comparisons separately.

d) Hypothesis testing as a supporting procedure: Recognizing the limitations of the small test data set and the conservativeness of traditional hypothesis testing procedures, this study treats these methods as supporting evidence, rather than the main driver for the conclusions. In addition to hypothesis testing, the study reports changes in proportions (accuracies), corresponding discordant pairs, each accompanied by Wilson score confidence intervals. Where appropriate, the author reports other supporting metrics, such as the absolute token cost of the optimization procedure. Conclusions are drawn from all of this, acknowledging limitations and remaining cautious about overstating the results.

K. Prompt Stability

In addition to the proportion of documents with correctly extracted target variables, the study also analyzes *stability* of each prompt by reporting the variability of its results in repeated runs. Let $\mathcal{D}$ denote the dataset (set of documents), with $|\mathcal{D}| = n$. The extraction procedure is repeated $k = 10$ times on the data set $\mathcal{D}$, under identical conditions with a fixed prompt p. Each inference is evaluated against the ground truth, yielding a per-document binary sequence:

$$x_d^{(p)} = \{x_{d,1}^{(p)}, \ldots, x_{d,k}^{(p)}\}, \quad x_{d,k}^{(p)} \in \{0, 1\},$$

Let $\mathrm{U}(\cdot)$ return the set of unique elements of its argument, and let $\mathbf{I}[\cdot]$ be the indicator function. Then, define *evaluation impersistence* for the prompt p on the data set $\mathcal{D}$ as the proportion of documents whose classification results vary between runs:

$$\mathrm{EIP}(p) \;=\; \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \mathbf{I}\Big(\big|\mathrm{U}(x_d^{(p)})\big| > 1\Big).$$

Independent of the binary evaluation, let

$$v_d^{(p)} = \{v_{d,1}^{(p)}, \ldots, v_{d,k}^{(p)}\}$$

denote the raw extracted values (before evaluation) returned in the same k runs. Define *response impersistence* as the proportion of documents whose raw outputs vary between runs:

$$\mathrm{RIP}(p) \;=\; \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \mathbf{I}\Big(\big|\mathrm{U}(v_d^{(p)})\big| > 1\Big).$$

Both metrics are reported as proportions of changed documents in a given dataset; higher values indicate greater variability (lower stability) of the prompt across repeated executions. The raw proportions (impersistence) are complemented with the 95%-CI calculated using the Wilson score, treating the proportion as the probability of a binomial distribution. The text of the study uses persistence and impersistence terms interchangeably (if the impersistence metric increases, the persistence of the model decreases). Additionally, the study reports the corresponding changes in the absolute number of altered documents.

L. Persistence testing across models

Most automated prompt optimization methods are inherently sensitive to the persistent behavior of the underlying LLM models, both for inference and feedback (if using the LLM-feedback component). However, this is often not the case with LLMs, and even under the exact same conditions and deterministic parameters, the response provided by the LLM might be different to the point of flipping the label. For example, this study observed that, in some cases, LLMs can extract the correct and expected value five times but fail five times in another instance.

Therefore, the model-wise persistence is analyzed before optimization, by observing the behavior of the Gemini Flash 2.0 and Gemini Flash 2.0 Lite models under the same experiment conditions during ten inference iterations on the scope 1 absolute emissions target variable with the corresponding initial prompt. The same impersistence metric from prompt stability is used, where persistence and impersistence are inverses of each other. As impersistence (the number of changed documents) increases, the persistence of the model or prompt in question decreases. Two metrics are reported: evaluation impersistence, which indicates how often the model's response evaluation changes across 10 iterations; and response impersistence, which indicates how often the model's response changes regardless of the resulting evaluation.

The impersistence across two models is reported only on Training and Validation sets (before any training), isolating the test set for the optimization part of the study. In contrast, only training and test set results are reported for prompt-induced persistence changes across optimization experiments.

M. Down-scaled runs

The research questions are answered based on the predetermined ProTeGi algorithm variant, referred to as ProTeGi-10it (which ran for 10 iterations).

However, the author also experiments with and reports other versions of ProTeGi, differing only in the number of iterations: ProTeGi-2it (ran for 2 iterations) and ProTeGi-5it (ran for 5 iterations). These algorithms are trained separately, each producing a unique prompt independent of each other. The down-scaled experiments were conducted after evaluating ProTeGi-10it and answering primary research questions, preventing any selection bias. For an overview of other parameters, see the Appendix.

N. Gradient verification

LLM feedback in general is an estimate of the real failure reason (gradient), rather than the actual gradient one would get in traditional backpropagation. The original ProTeGi [9] method addresses this problem by exploring multiple prompts and gradients simultaneously, potentially inflating costs and lowering performance (due to less guided exploration).

This paper proposes an alternative solution by introducing a decision gate following LLM feedback, which drops the said feedback if it does not improve the performance on the data it was generated from. The algorithm is built on the assumption that the LLM-proposed gradient generated for incorrect examples is an accurate estimate of the real gradient if it corrects the said examples. This may not always hold, highlighting the need for a robust parent optimization process that can handle this limitation. For instance, iterative methods such as ProTeGi are expected to work well with the proposed procedure, while single-step optimization methods could fail.

The underlying principle is defined as "LLM-feedback generated from an incorrect piece of data is less valuable if it does not cause a change in the correctness of the said data when applied". The study refers to the concept as GradV (which stands for Gradient Verification). Still, the idea can be applied more generally to any prompt optimization methodology that relies on LLM feedback. The proposed verification procedure is formally described in Algorithm 1.

The process begins with a data point d, a prompt p, and a textual gradient g, where g is a feedback piece proposed by an LLM, aiming to estimate the fundamental error that the prompt p introduces in the data point d. Further, three abstract functions are defined and executed in sequence:

- MODIFY - A function that modifies the incoming prompt p in the direction of the gradient g, creating a new prompt p'. In other words, the function applies LLM feedback to the prompt, producing a new candidate prompt. In a broader concept of LLM-feedback-based prompt optimization, the function can take different forms, such as complete rewrites (as in ProTeGi), appends (as in AutoHint [20] and the newly proposed method), or anything in between.
- INFER - A function (shortened from Inference), which takes in the aforementioned data point d and the prompt p', performing the task for which the prompt is optimized, producing a predicted value/label y.
- METRIC - A function that takes the label y and the ground truth $y^\star$ or the data point d, returning a binary classification result c, determining whether

the answer is correct. For qualitative tasks, the function can be extended towards determining if the answer is good enough.

If c is 1, the LLM-generated textual gradient (or feedback in a more general sense) is probably a good estimator of the actual reason why the prompt p failed on the data point d. This allows the parent optimization algorithm to focus exploration in the direction of the said validated gradient. Otherwise, the gradient is likely not a good estimator and can be discarded.

Algorithm 1 GRADIENTVERIFICATION

Require: textual gradient g, data point d, prompt p, ground truth $y^\star$; functions MODIFY, INFER, METRIC
Ensure: either (g, p') or no output
1: $p' \leftarrow \text{MODIFY}(p, g)$
2: $y \leftarrow \text{INFER}(p', d)$
3: $c \leftarrow \text{METRIC}(y, y^\star)$
4: **if** $c = 1$ **then**
5: **return** (g, p')
6: **else**
7: **return** ▷ halt without output
8: **end if**

Due to its flexibility, the proposed gradient verification procedure can accommodate various prompt optimization methods with a present LLM-feedback component. The GradV procedure is not limited to a single data point either; defining the metric function to support a dataset batch rather than a single point allows for verifying gradients produced from a batch of data on the said batch. However, in the case of algorithms that explore multiple new prompts per gradient, inducing the quality of the said gradient is more complicated. In such instances, verification could be performed on candidate prompts rather than gradients. Still, the proposed concept remains the same: There is no need to explore gradients and candidate prompts generated on incorrect data if they do not impact performance on the said data.

A special note must be made regarding the persistence of the underlying LLM model, as the same conditions do not always guarantee the same output, as observed in this study (see the Results section). This could potentially cause a random label flip in the label/output returned from the inference function, leading the metric function to decide that the correctness of the underlying example has changed, effectively propagating inaccurate gradients or prompts.

To address this, the study extends the decision gate to conduct repeated inferences and only allows the gradient/prompt to pass if the aggregated score exceeds the threshold. For example, the function INFER evaluates the same document multiple times, producing multiple labels y. Sequentially, the function METRIC compares all the generated labels y with the truth $y^\star$, determining the correctness c based on the average score set against the threshold. In all following implementations of GradV (PE and ProTeGi-5it), each document is evaluated twice, and the feedback is only validated if both inferences are correct. Stricter criteria might still be beneficial; the author compromises on two to filter a proportion of label flips while still keeping costs down.

O. Proof-of-concept for Prompt Enrichment (PE) method using gradient verification procedure

The study proposes a brief and concise prompt optimization method based solely on LLM feedback with gradient verification. In contrast to ProTeGi, the gradient is treated as an instruction additive to the prompt, addressing the error observed by the LLM. This bypasses the need for rewrites, allowing the algorithm to directly infer the accuracy of the gradient estimator from the performance of the modified prompt. In other words, the generated gradient is directly formulated as an instruction to fix the problem. To distinguish from a textual gradient, this instruction piece is named r, short for "reason". For a complete overview of the algorithm workflow, see Algorithm 2.

Algorithm 2 PROMPTENRICHMENT

Require: dataset D, starting prompt p_0, rounds/iterations k, reasons per round m; functions CLASSIFY, GENREASONS, POOL, FILTER, CONCAT
Ensure: final prompt $p^\star$, unfixed examples $BadEx$
1: $(-D, +D) \leftarrow \text{CLASSIFY}(D, p_0)$
2: $(R, BadEx) \leftarrow \text{GENREASONS}(p_0, -D, k, m)$
3: $CombReasons \leftarrow \text{POOL}(R)$
4: $SelectedReasons \leftarrow \text{FILTER}(CombReasons, p_0, D)$
5: $p^\star \leftarrow \text{CONCAT}(p_0, SelectedReasons)$
6: **return** ($p^\star$, BadEx) ▷ Final prompt

The overarching principle of the proposed algorithm is opposite to ProTeGi, attempting to capture and validate textual descriptions of all inference errors present in the training dataset, rather than nudging individual prompts. The study formulates the textual description in terms of the reasons introduced earlier.

Each of the reasons is obtained independently for each incorrect example; furthermore, they are independent of the primary prompt and interchangeable, where any sequence of the reasons could be appended to the starting prompt one by one. Naturally, this imposes a constraint on the starting prompt to capture the correct task descrip-

tion, which only needs to be extended with instructions to address problems observed in the training dataset.

At the end of the optimization, the process returns a list of validated reasons describing the errors observed in the dataset, as well as a list of "bad examples" - examples for which a valid reason could not be found within the specified number of iterations. The latter shows cases where the model struggles, irrespective of the explored reasons, providing valuable feedback for human engineers. In addition, each reason is evaluated with two copies of the parent example and is only considered correct if both are correct, shielding from LLM persistence problems.

The proposed algorithm could be used for both transparent automated prompt optimization and the generation of the aforementioned textual descriptions of the errors in the dataset. The latter might be valuable for human engineers implementing LLM-based workflows.

1) Generating Reasons: The algorithm begins by splitting the dataset into two sets: correct and incorrect examples. The latter set is then used to repeatedly generate and check gradients per example, returning a list of validated reasons and bad examples when finished (see Algorithm 3). The output list of reasons can already be used to examine errors produced by each example by analyzing the respective fixes. However, at this stage, each reason is generated independently of the others, and the impact on the rest of the dataset is unclear. To evaluate the impact while minimizing costs, the list is optimized by removing redundancy through grouping and merging similar reasons. This part of the algorithm is referred to as pooling.

2) Pooling: In the suggested proof-of-concept, the pooling (see Algorithm 4) is entirely delegated to the LLM, simplifying the workflow. First, LLM performs pairwise comparison of reasons, returning a binary similarity classification for each pair (similar/dissimilar). Secondly, all similar instructions are then grouped, inferring that if an instruction appears in two pairs as similar to two other distinct instructions, then all three are similar. Although this might not always hold, the study formulates the said assumption to escape a complex optimization problem. Third, LLM merges all reasons from the same similarity group, resulting in a list of instructions with lower redundancy. The said list could be further analyzed for similarity or verified against the connected examples again, but this is beyond the scope of the proposed proof of concept. Additionally, a production-ready framework may want to employ more robust similarity methods, such as cosine similarity of embedding vectors.

3) Filtering: The proof-of-concept algorithm produces the optimized prompt following a naive approach of appending a set of instructions to the original prompt,

Algorithm 3 GENREASONS

Require: incorrect set $-D$, starting prompt p_0, rounds/iterations k, reasons per round m; functions CONCAT, INFER, METRIC, REASON

Ensure: reasons $Reasons$, unfixable examples $BadEx$

$Reasons \leftarrow \emptyset$
$BadEx \leftarrow \emptyset$
for all $d \in -D$ **do** ▷ parallelized
  $fixed \leftarrow$ false
  for $i = 1$ **to** k **do**
    $R \leftarrow \text{REASON}(p_0, d, m)$ ▷ Get m reasons (instructions) fixing the problem
    for all $r \in R$ **do** ▷ parallelized
      $p' \leftarrow \text{CONCAT}(p_0, r)$
      $y \leftarrow \text{INFER}(p', d)$
      if $\text{METRIC}(d, y) = 1$ **then**
        append r to $Reasons$; $fixed \leftarrow$ true
        break
      end if
    end for
  end for
  if not $fixed$ **then**
    append d to $BadEx$
  end if
end for
return $(Reasons, BadEx)$

Algorithm 4 POOL

Require: reasons R; functions SIM, SOLVESIMILAR, COMBINE

Ensure: combined reasons $CombReasons$

$S \leftarrow \{(i, j, \text{SIM}(r_i, r_j)) : 1 \leq i < j\}$ ▷ pairwise similarity
$P \leftarrow \text{SOLVESIMILAR}(R, S)$ ▷ partition into groups
$CombReasons \leftarrow \emptyset$
for all $\mathcal{S} \in P$ **do**
  $r \leftarrow \text{COMBINE}(p_0, \mathcal{S}, D)$
  append r to $CombReasons$
end for
return $CombReasons$

simplifying the workflow; however, this hinges on a strong assumption of independence between the grouped reasons. For example, as no interaction between different components is explored, a pair of two instructions that both improve performance might otherwise lower it when combined.

The study defines the independence assumption as two conditions that must be met: no reasons added to the final prompt contradict each other, and no reasons are similar. The latter is expected to have a minor impact on performance, as adding two similar instructions to the

prompt is unlikely to significantly lower performance compared to each instruction appended separately. In contrast, contradicting instructions could directly reduce performance, especially considering that they are produced on a per-example level and, hence, are sensitive to outliers. Addressing the contradictory component of the independence assumption, a procedure similar to similarity classification is performed, this time using LLM to determine contradictory pairs (see Algorithm 5).

Each of the instructions in the reduced list is evaluated independently of the others (by appending to the starting prompt) on the entire training dataset. Instructions that reduce performance compared to the starting prompt are eliminated. As evaluation of the whole training data set is one of the more expensive algorithm components, one could attempt to decrease the costs by evaluating on a subset of the data (for example, using bandit evaluation). Sequentially, LLM detects contradicting pairs by performing pairwise similarity classification.

Having both raw performance for each instruction and their pairwise contradiction classifications, a classical linear programming algorithm is then utilized, collecting a final set of non-contradicting instructions while maximizing the score on the dataset (by assuming additive score properties). Although the final list of instructions is not guaranteed to be non-contradicting or non-overlapping, it's estimated to be so by an LLM.

Algorithm 5 FILTER

Require: combined reasons $CombReasons$, starting prompt p_0, dataset D; functions INFER, METRIC, CONCAT, CONTR, SOLVESELECTION

Ensure: selected reasons $SelectedReasons$

Def $\mathrm{S}(p; D) \leftarrow \frac{1}{|D|} \sum_{d \in D} \textsc{Metric}(d, \textsc{Infer}(p, d))$
Def $\mathrm{C}(p_0, r) \leftarrow \textsc{Concat}(p_0, r)$
$s_0 \leftarrow \mathrm{S}(p_0; D)$
$T \leftarrow \emptyset$ ▷ cache of scores
for all $r \in CombReasons$ **do**
    $T[r] \leftarrow \mathrm{S}(\mathrm{C}(p_0, r); D)$
end for
$R_{keep} \leftarrow \{r \in T : T[r] \geq b\}$
for all $r \in R_{keep}$ **do**
    $w_r \leftarrow T[r] - s_0$
end for
$E_{con} \leftarrow \{\{i, j\} \subseteq R_{keep} : \textsc{Contr}(r_i, r_j) = 1\}$
$X^\star \leftarrow \textsc{SolveSelection}(R_{keep}, E_{con}, \{w_r\})$
$SelectedReasons \leftarrow \{r_i : i \in X^\star\}$
return $SelectedReasons$

4) Special notes: A special note needs to be made regarding the two hyperparameters: the number of iterations per example and the number of reasons generated during each iteration. The former is set to 5 and the latter to 3, approximately matching the costs of the corresponding ProTeGi variant (ProTeGi-2it) for comparison purposes. Longer runs were attempted, but diminishing returns were observed, highlighting the limitations of the naive prompt optimization algorithm used as the proof of concept.

In the proposed proof-of-concept, the workflow is simplified and iterative exploration is not present, which may potentially limit improvement and decrease the generalizability of the results. Further, as the reasons are produced per document, the risk of overfitting becomes apparent. The algorithm mitigates this by employing pooling, evaluation, and contradiction analysis. However, this might not be the most optimal solution and poses its own problems, mainly LLM hallucinations in the absence of iterativeness.

Addressing this, the proposed gradient verification procedure is also tested as an extension of the recognized ProTeGi prompt optimization framework.

P. ProTeGi with gradient verification

This study combines the proposed gradient verification procedure with the ProTeGi algorithm to harness the advantages of both: iterative and exploratory search over multiple prompts provided by ProTeGi, and the ability to identify more prominent prompts at a low cost, as provided by the Gradient Verification procedure. The proposed extension to the ProTeGi algorithm is expected to guide the learning process more effectively, focusing the attention and resources provided by the algorithm solely on the promising part of the prompt space.

The procedure is injected into the expansion step following the generation of candidate prompts. As the duality of the gradient-candidate prompt pair is present in ProTeGi, contrasting with the proposed proof-of-concept algorithm, the verification is applied directly to the produced candidate prompt rather than the gradient. Furthermore, no repeated gradient generation is included; the procedure is introduced solely as a decision gate, determining whether each of the proposed prompt candidates should be included in the set of new candidates submitted to the UCB evaluation. This modification enables the reduction of evaluation costs while focusing the algorithm on exploring the promising part of the prompt space, rather than the entire space.

Due to the additional sequentiality and processing time required by the gradient verification, the study limits ProTeGi to a 5-it version, following the hypothesis of a better guided search and consequently faster convergence. The author expects similar performance to the full 10-it version of the algorithm at around half the cost.

Explicitly addressing the persistence problems, the gradient verification is performed on two copies of the same document, only letting those that make both correct through.

Q. Train vs. Test set distinction

The study reports all metrics (including p-values) for both Train and Test sets for completeness purposes (and due to limited datasets); however, conclusions are drawn primarily from the test set. Furthermore, the hypothesis testing procedure considers exclusively p-values from the test sets and therefore applies the Holm-Bonferroni correction only to the test set.

IV. Results

In the results section, all hypothesis tests refer to the ProTeGi-10it variant, as specified in the methodology; other variants are reported descriptively only. Furthermore, all hypothesis tests are performed on the test set only (n = 49), as specified in the methodology.

A. Persistence across Gemini Models

The author observed evaluation changes in $\approx$ **16%** of the documents run through Gemini Flash 2.0 (see Table V). The change in responses is even greater, where $\approx$ **24%** of the data points incurred at least one change in response during the 10 iterations (see Table VI).

Gemini Flash-Lite 2.0 model displays higher persistence (lower impersistence) in both evaluations V and responses VI on the training dataset. In contrast, persistence on the Validation dataset for the Lite model is lower. However, the difference between the two models on the Validation dataset is minor (an additional 1 and 2 documents are changed for the Lite Model across both persistence categories), and the Validation dataset is relatively small, making this inconclusive. In contrast, on the larger training dataset, the change is more pronounced, with the number of changed documents dropping by more than a factor of two for Gemini Lite in both persistence categories.

Although the behavior of the Flash 2.0 model can impede the learning process, and the Lite model shows better persistence, the author selects Flash 2.0 for optimization and inference to prevent the potential performance drop associated with the Lite version. The study also attempts to mitigate the persistence effect by widening the training process with more iterations and a larger beam size.

B. Hypothesis I: Optimized prompt is an improvement over the initial prompt

The automatically optimized prompt presents a consistent improvement over the initial prompt on the test data set: S1 increased from **53%** to **80%** (by **27pp**); S3a from **61%** to **65%** (by **4pp**); Rp from **84%** to **94%** (by **10pp**) (see Table VII).

Model	Dataset	Change count	EIP (95% CI)
Flash	Train	11/64	0.17 (0.10, 0.28)
Flash	Val	4/28	0.14 (0.06, 0.31)
Flash	Aggregated	15/92	0.16 (0.10, 0.25)
Lite	Train	4/64	0.06 (0.02, 0.15)
Lite	Val	5/28	0.18 (0.08, 0.36)
Lite	Aggregated	9/92	0.10 (0.05, 0.18)

TABLE V: Persistence in evaluations. Reported are absolute counts and evaluation impersistence metric (binomial proportions (EIP)) with 95% Wilson confidence intervals.

Model	Data	Change count	RIP (95% CI)
Flash	Train	18/64	0.28 (0.19, 0.40)
Flash	Val	4/28	0.14 (0.06, 0.31)
Flash	Aggregated	22/92	0.24 (0.16, 0.34)
Lite	Train	7/64	0.11 (0.05, 0.21)
Lite	Val	6/28	0.21 (0.10, 0.40)
Lite	Aggregated	13/92	0.14 (0.08, 0.23)

TABLE VI: Persistence in model responses. Reported are absolute counts and response impersistence metric (binomial proportions (RIP)) with 95% Wilson confidence intervals.

Although training was carried out under the same hyperparameters and training costs are similar IX, a varying level of response to training is observed. For example, S3a (assurance of scope 3) improved only by 4pp compared to the baseline. This is in connection with the S3a potentially displaying a higher level of task complexity. While S1 is often presented as a simple number to be extracted, S3a must be inferred from context. Furthermore, identifying S3a typically involves multiple steps, including determining whether the emission is mentioned, whether it is validated, and whether an independent entity performs the validation (see the manual prompt in the Appendix). In contrast, S1 is more straightforward, where many of the observed errors were due to incorrect units and conversion errors.

Delving into the prompts themselves (see the Appendix), the optimized prompt for S3a is merely a rephrasing of the starting prompt with additional details; in contrast, S1 includes many easily recognizable components, such as instructions for conversion and handling of unit prefixes, omitting projections, and so on. The reporting period also saw an improvement of 10 points from the optimized prompt, with a set of distinguishable instructions appearing in the prompt, similar to S1. However, the improvement is likely smaller than in S1 due to the substantially higher starting accuracy of 84%.

Following the hypothesis testing procedure with Holm-Bonferroni, the set of test p-values from Table VII is ordered in ascending order (0.0005, 0.0313, 0.4194) and

tested against significance levels scaled by the stepwise number of variables (0.05/3, 0.05/2, 0.05/1). Only the first p-value (0.0005) passes the test, allowing rejection of the null hypothesis for S1 and denying rejection for S3a and Rp. This means that only the improvement in S1 is statistically significant at the 0.05 significance level when tested on discordant pairs. Rp is significant without correction (p = 0.0313), but not after Holm-Bonferroni (a = 0.025 at step 2). However, the results are sensitive where the number of discordant pairs is small; the tests can be imprecise and underpowered, particularly for Rp (15 pairs for S1, 24 for S3a, 5 for Rp).

The 95% confidence intervals for the proportion of discordant pairs in favor of an optimized prompt also show a similar result (see Table VII). For S1, the lower bound is 20pp above the random 50-50 split expected under the null hypothesis. For Rp, the lower bound of the confidence interval is also higher than 50%, but only by 7pp, which explains why it barely missed the corrected significance level. In contrast, the lower bound for S3a lies 15pp below 50%, and the discordant odds in favor of an optimized prompt are 1.18.

The varying level of improvement highlights the task dependence of the prompt optimization algorithm, where simpler tasks can be effectively automated, giving a significant boost to accuracy at a certain cost. In contrast, the improvement for more complex tasks is minor and often statistically insignificant given small datasets. Effectively, this highlights the limitations of using LLMs for specific information extraction tasks, where deeper contextual and domain-specific knowledge analysis is required.

The total training cost and runtime statistics are presented in Table IX. Although the number is substantial, the cost is inflated by the size of the documents. A solution to the cost problem could lie in a prior retrieval system, which could decrease costs by selecting only a relevant part of the document (reducing the mean of 150 I to 10 pages would cut the cost by $\sim$15 times).

Overall, the study observes a clear improvement in the optimized prompt compared to the starting prompt; however, this improvement is not always statistically significant. Considering the small power of the study (due to small datasets) and statistics observed in Table VII (and discussed above), the author draws a cautious conclusion in favor of automated prompt optimization: Automated prompt optimization (in the form of ProTeGi) can improve the performance of information extraction workflows; however, the size and significance of the improvement are task-specific and the optimization framework does not guarantee a significant universal improvement.

C. Hypothesis II: The performance of auto-optimized prompts is significantly different from that of manually engineered prompts

The manual prompt shows better accuracy than the optimized one on S1 (see Table VIII), but the difference is minor. The optimized prompt beats the manual one on the Rp target by **10pp**, while losing by **8pp** on S3a. Overall, the difference is inconsistent between variables, where optimized prompts show similar or better performance than manual prompts on S1 and Rp, while losing on S3a. Upon closer examination, the optimized prompt for S3a is essentially a paraphrase of the initial prompt. In contrast, the manual prompt includes a list of detailed instructions that cover edge cases (see the Appendix). This demonstrates the potential limitations of LLM-feedback prompt optimization procedures, where improvements on certain variables can be minor and inconsistent.

For all three target variables on the test set, the two-sided p-value for a binomial paired test on discordant pairs is higher than the threshold of 0.05; the results remain non-significant after Holm-Bonferroni correction; however, since the number of discordant pairs is small (3 for S1, 16 for S3a, 5 for Rp), these tests are underpowered with imprecise estimates. A further formal test, such as non-inferiority or equivalence analysis, would require more data and a predefined margin.

Overall, the evidence is consistent with similar performance between auto-optimized and manually engineered prompts on some target variables, based on similar proportions (accuracies) and overlapping corresponding confidence intervals. However, determining statistical significance would require more data and a further non-inferiority/equality test. Furthermore, a moderation by task complexity is evident, where, for S3a specifically, the manual prompt outperforms the optimized one, while the optimized shows a slightly better performance than the starting one.

At the same time, the automated prompt optimization procedure enhances accuracy compared to the initial prompt. Consequently, while acknowledging the above significance problem, the procedure can be considered a practical approach to obtaining competitive prompts for information extraction tasks, especially when the number of prompts required is high, making manual prompt engineering infeasible.

Another dimension of automated prompt optimization is the costs (see Table IX). The author further experiments with different versions of the ProTeGi algorithm by customizing the number of iterations while constraining other variables, resulting in scaled-down versions of the

Var	Data	Starting (95%-CI)	Optimized (95%-CI)	Δ	Opt wins	St wins	D. odds	p (95%-CI)	H1: p1
S1	Train	0.47 (0.35, 0.59)	0.75 (0.63, 0.84)	0.28	21	3	7.00	0.88 (0.69, 0.96)	0.0001
	Test	0.53 (0.39, 0.66)	0.80 (0.66, 0.89)	0.27	14	1	14.00	0.93 (0.70, 0.99)	0.0005
S3a	Train	0.50 (0.38, 0.62)	0.72 (0.60, 0.81)	0.22	24	10	2.40	0.71 (0.54, 0.83)	0.0122
	Test	0.61 (0.47, 0.74)	0.65 (0.51, 0.77)	0.04	13	11	1.18	0.54 (0.35, 0.72)	0.4194
Rp	Train	0.75 (0.63, 0.84)	0.84 (0.74, 0.91)	0.09	8	2	4.00	0.80 (0.49, 0.94)	0.0547
	Test	0.84 (0.71, 0.91)	0.94 (0.83, 0.98)	0.10	5	0	–	1.00 (0.57, 1.00)	0.0313

TABLE VII: Proportion of correctly classified items by target variable and dataset for Auto-Optimized vs Starting prompts. Var refers to the target variable, Starting/Optimized - to Starting/Auto-Optimized prompt proportions/accuracies followed by respective confidence intervals in parentheses. Δ (Opt-Start) shows the absolute difference between optimized and starting prompts' performance. Opt wins refers to cases where the auto-optimized prompt is correct and starting is not; Start wins - cases where starting is correct and the auto-optimized is not; D. odds - ratio of discordant pairs. Ratio is undefined when the denominator = 0. p refers to the proportion of discordant pairs favoring the optimized prompt with Wilson's 95% confidence interval. H1 refers to a one-tailed binomial exact test of whether the number of discordant pairs is significantly in favor of an improved prompt.

Var	Data	Manual (95%-CI)	Optimized (95%-CI)	Δ	Opt wins	Man wins	D. odds	p (95%-CI)	H2: p2
S1	Train	0.70 (0.58, 0.80)	0.75 (0.63, 0.84)	0.05	5	2	2.50	0.71 (0.36, 0.92)	0.4531
	Test	0.82 (0.69, 0.90)	0.80 (0.66, 0.89)	0.02	1	2	0.50	0.33 (0.06, 0.79)	1.0000
S3a	Train	0.84 (0.74, 0.91)	0.72 (0.60, 0.81)	0.12	3	11	0.27	0.21 (0.08, 0.48)	0.0574
	Test	0.73 (0.60, 0.84)	0.65 (0.51, 0.77)	0.08	6	10	0.60	0.38 (0.18, 0.61)	0.4545
Rp	Train	0.75 (0.63, 0.84)	0.84 (0.74, 0.91)	0.09	10	4	2.50	0.71 (0.45, 0.88)	0.1796
	Test	0.84 (0.71, 0.91)	0.94 (0.83, 0.98)	0.10	5	0	–	1.00 (0.57, 1.00)	0.0625

TABLE VIII: Proportion of correctly classified items by target variable and dataset for Auto-Optimized vs Manual prompts. Var refers to the target variable, Optimized/Manual - to Auto-Optimized/Manually-written prompt proportions/accuracies followed by respective confidence intervals in parentheses. Δ (Opt-Man) represents the absolute difference between the performance of optimized and manual prompts. Opt wins refers to cases where the auto-optimized prompt is correct and manual is not; Man wins - cases where manual is correct and auto-optimized is not; D. odds - ratio of discordant pairs (undefined when denominator = 0). p refers to the proportion of discordant cases favoring optimized prompt with Wilson 95% confidence interval (undefined when no discordant pairs). H2 refers to a two-tailed exact binomial test on whether discordant pairs differ between optimized and manual.

Process	S1			S3a			Rp		
	Tokens	API Calls	Uptime (s)	Tokens	API Calls	Uptime (s)	Tokens	API Calls	Uptime (s)
Inference	191 453 161	4 646	169 836.71	190 811 273	4 626	148 423.28	192 208 962	4 676	154 945.27
Feedback	6 695 743	196	3 833.86	2 935 259	190	1 673.90	3 247 460	216	1 888.89

TABLE IX: Training-cost summary for the three target variables. Tokens = total input + output tokens processed; API Calls = number of processed requests sent to the Gemini endpoints (does not include failed requests); API Uptime = cumulative API processing time in seconds (time spent waiting for all API requests to process, does not include failed requests); Inference = information extraction part of the process; Feedback = "Learning component" of the optimization process.

optimization algorithm.

D. Variability of the results

All stability metrics are reported for 10 identical runs at a temperature of 0.01 with vendor defaults for other parameters, as explained in the methodology.

On test set, optimized prompts are more stable than starting for S1 (RIP **22%→8%**, EIP **14%→4%**) and Rp (RIP **8%→0%**, EIP **6%→0%**), but less stable on S3a (RIP **6%→12%**, EIP **6%→12%**) (see Tables X, XI).

Compared to manual prompts, the stability is similar on S1 (RIP **4%→8%**, EIP **2%→4%**) and S3a (RIP **10%→12%**, EIP **8%→12%**). In contrast, the optimized prompt is more stable on Rp (RIP **22%→0%**, EIP **16%→0%**). The accuracy in 10 runs varies modestly, with a standard deviation of 0-3pp (see Table XII).

Although persistence experiments are conducted while controlling the stochasticity of the underlying LLM (by setting the temperature near zero), up to 22% of documents experienced a change in responses during 10 iterations on some variables, highlighting robust-

ness concerns in the performance of both LLM-backed information extraction systems and their performance evaluation.

Furthermore, as the accuracy across runs varies modestly (standard deviation of 0-3pp), the effect might be worth accounting for in full-scale prompt optimization and information extraction tasks. This becomes particularly relevant when pipelines are based on a single-inference run, as the results can differ even when controlling for stochasticity by setting near-zero temperature.

E. Down-scaled optimization algorithms

For most target variables, a steady growth in prompt accuracy is observed, coinciding with the number of training iterations (see Table XIII). For S1, ProTeGi-5it captured most of the gains with **23pp** improvement (**53%**→**76%**), while ProTeGi-10it version only added a **4pp** extra (**76%**→**80%**).

ProTeGi-2it quickly reached a peak for Rp, improving by **10pp** over the initial prompt (**84%**→**94%**); ProTeGi-5it showed a smaller improvement over the baseline (**84%**→**88%**), while ProTeGi-10it also reached **94%**.

In contrast, the accuracy of S3a only peaked for ProTeGi-10it (**61%**→**65%**) with a minor improvement of **4pp** over the initial prompt. This highlights the complexity of the S3a target and diversity of the dataset, where the improvement on the train set is already apparent for ProTeGi-2it, but is not generalizable to the test dataset, either overfitting or learning the specifics not present in the test set due to a small sample size and/or outliers in the data.

Overall, the experiment demonstrates that adjusting the iteration parameter properly can make optimization more resource-efficient. However, different iteration versions exhibit task-specific behavior, where ProTeGi-5it in S1 achieved nearly the same results as ProTeGi-10it, but not in S3a. ProTeGi-2it was sufficient to reach the highest observed score on Rp.

F. Prompt Enrichment (PE) concept results

Multiple parameter configurations of the proposed method (PE) were experimented with; however, diminishing returns beyond the default first run of the ∼50 million token version were observed in the train and validation datasets, likely due to the poor scalability of the grouping and filtering layers and the absence of an iterative optimization component. However, during the aforementioned starting ∼50 million token runs, a steady albeit slight improvement was observed across categories.

PE improves over the starting prompt by: **23pp** on S1 (**53%**→**76%**); **2pp** on S3a (**61%**→**63%**); **4pp** on Rp (**84%**→**88%**) (see Table XIV). The PE produced prompts match the performance of ProTeGi-5it at half the cost (∼ 50M vs. ∼ 100M) (see Table XIV). Furthermore, since PE costs depend on the proportion of documents initially incorrect, the cost for scenarios with higher accuracy (Rp) is lower (Rp: ∼ 30M vs. S1: ∼ 50M).

Qualitatively, the algorithm provides a list of clear rationales for errors in the data set, which could prove a valuable asset for transparency and downstream optimization (see the Appendix). The rationales produced appear to be informative of the underlying errors, mentioning similar reasons observed in the optimized prompts of ProTeGi.

Based on the above observations and the small dataset, the study draws a cautious conclusion that the proposed proof-of-concept algorithm and the underlying gradient verification procedure could be utilized to produce an improvement over the initial prompts in the information extraction use case, competing with ProTeGi-5it while being more cost-effective. However, more research and improvements are needed on the proposed concept, particularly in addressing the pooling and filtering layers. The evidence observed is consistent with the conclusion; however, the comparison can be imprecise due to small dataset sizes and minor effects observed.

G. ProTeGi with gradient verification

Auto-optimized prompts produced by ProTeGi-5it, enhanced with the GradV procedure, also demonstrate a clear improvement over the starting prompt's accuracy: S1 improves by **27pp** (**53%**→**80%**), S3a by **10pp** (**61%**→**71%**), and Rp by **8pp** (**84%**→**92%**) (see Table XV). The proposed algorithm rivals ProTeGi-10it on S1 and Rp, while beating it on S3a by **6pp** (**65%**→**71%**). Compared to standard ProTeGi-5it, the GradV enhanced version shows a **4pp** improvement on both S1 and Rp targets (**76%**→**80%**, **88%**→**92%**); and a **14pp** uplift on S3a (**57%**→**71%**). Simultaneously, it reduces the costs by 0-30% over basic ProTeGi-5it and 50% over ProTeGi-10it.

The improvement observed on S3a is particularly notable, showing the highest increase of all algorithms explored on arguably the most complex target (displaying the lowest sensitivity to prompt optimization). Qualitatively, the prompt explicitly includes the Scope 3 emissions term and asks not to rely on the term itself, but to seek assurance verification beyond Scopes 1 and 2 (See the Appendix). This detail is unique and has not been observed in other optimized prompts.

The empirical and qualitative evidence reported is consistent with the intuition behind the GradV gateway as a facilitator of a more effective exploration of the

Var	Data	Starting (Count)	RIP (95%-CI)	Optimized (Count)	RIP (95%-CI)	Manual (Count)	RIP (95%-CI)
S1	Train	13/64	0.20 (0.12, 0.32)	8/64	0.12 (0.06, 0.23)	9/64	0.14 (0.08, 0.25)
	Test	11/49	0.22 (0.13, 0.36)	4/49	0.08 (0.03, 0.19)	2/49	0.04 (0.01, 0.14)
S3a	Train	9/64	0.14 (0.08, 0.25)	1/64	0.02 (0.00, 0.08)	5/64	0.08 (0.03, 0.17)
	Test	3/49	0.06 (0.02, 0.17)	6/49	0.12 (0.06, 0.24)	5/49	0.10 (0.04, 0.22)
Rp	Train	6/64	0.09 (0.04, 0.19)	3/64	0.05 (0.02, 0.13)	16/64	0.25 (0.16, 0.37)
	Test	4/49	0.08 (0.03, 0.19)	0/49	0.00 (0.00, 0.07)	11/49	0.22 (0.13, 0.36)

TABLE X: Response persistence per prompt across 10 iterations for S1, S3a, and Rp. Reported are absolute counts and Response Impersistence metric (binomial proportions (RIP)) with 95% Wilson confidence intervals.

Var	Data	Starting (Count)	EIP (95%-CI)	Optimized (Count)	EIP (95%-CI)	Manual (Count)	EIP (95%-CI)
S1	Train	8/64	0.12 (0.06, 0.23)	5/64	0.08 (0.03, 0.17)	7/64	0.11 (0.05, 0.21)
	Test	7/49	0.14 (0.07, 0.27)	2/49	0.04 (0.01, 0.14)	1/49	0.02 (0.00, 0.11)
S3a	Train	8/64	0.12 (0.06, 0.23)	1/64	0.02 (0.00, 0.08)	4/64	0.06 (0.02, 0.15)
	Test	3/49	0.06 (0.02, 0.17)	6/49	0.12 (0.06, 0.24)	4/49	0.08 (0.03, 0.19)
Rp	Train	4/64	0.06 (0.02, 0.15)	3/64	0.05 (0.02, 0.13)	14/64	0.22 (0.14, 0.33)
	Test	3/49	0.06 (0.02, 0.17)	0/49	0.00 (0.00, 0.07)	8/49	0.16 (0.09, 0.29)

TABLE XI: Evaluation persistence per prompt across 10 iterations for S1, S3a, and Rp. Reported are absolute counts and Evaluation Impersistence (binomial proportions (EIP)) with 95% Wilson confidence intervals.

Var	Data	Starting (Mean, Std)	Optimized (Mean, Std)	Manual (Mean, Std)
S1	Train	0.494, 0.020	0.736, 0.014	0.686, 0.020
	Test	0.537, 0.017	0.786, 0.011	0.820, 0.009
S3a	Train	0.516, 0.023	0.717, 0.005	0.853, 0.008
	Test	0.600, 0.014	0.665, 0.017	0.727, 0.014
Rp	Train	0.806, 0.015	0.856, 0.010	0.748, 0.031
	Test	0.839, 0.020	0.939, 0.000	0.820, 0.027

TABLE XII: Accuracy/proportion of correct documents across 10 iterations (performed for persistence experiment) for S1, S3a, and Rp on the three prompts. Reported are sample means with sample standard deviations.

Var	Data	Starting	ProTeGi-2it		ProTeGi-5it		ProTeGi-10it		Manual
			Acc.	Tokens	Acc.	Tokens	Acc.	Tokens	
S1	Train	0.47 (0.35,0.59)	0.67 (0.55,0.77)	∼50M	0.78 (0.67,0.86)	∼100M	0.75 (0.63,0.84)	∼200M	0.70 (0.58,0.80)
	Test	0.53 (0.39,0.66)	0.61 (0.47,0.74)	-	0.76 (0.62,0.85)	-	0.80 (0.66,0.89)	-	0.82 (0.69,0.90)
S3a	Train	0.50 (0.38,0.62)	0.62 (0.50,0.73)	∼50M	0.70 (0.58,0.80)	∼100M	0.72 (0.60,0.81)	∼200M	0.84 (0.74,0.91)
	Test	0.61 (0.47,0.74)	0.59 (0.45,0.72)	-	0.57 (0.43,0.70)	-	0.65 (0.51,0.77)	-	0.73 (0.60,0.84)
Rp	Train	0.75 (0.63,0.84)	0.88 (0.77,0.94)	∼50M	0.89 (0.79,0.95)	∼100M	0.84 (0.74,0.91)	∼200M	0.75 (0.63,0.84)
	Test	0.84 (0.71,0.91)	0.94 (0.83,0.98)	-	0.88 (0.76,0.94)	-	0.94 (0.83,0.98)	-	0.84 (0.71,0.91)

TABLE XIII: Accuracies/Proportions with 95% Wilson confidence intervals and approximate token cost for training (rounded to the upper 10M) for different versions of ProTeGi (2, 5, 10 iterations) across target variables (S1, S3a, Rp) and datasets (Train, Test). Starting and Manual prompts are shown without costs.

Var	Data	Starting	ProTeGi-2it		ProTeGi-5it		PE		Manual
			Acc.	Tokens	Acc.	Tokens	Acc.	Tokens	
S1	Train	0.47 (0.35,0.59)	0.67 (0.55,0.77)	∼50M	0.78 (0.67,0.86)	∼100M	0.70 (0.58,0.80)	∼50M	0.70 (0.58,0.80)
	Test	0.53 (0.39,0.66)	0.61 (0.47,0.74)	-	0.76 (0.62,0.85)	-	0.76 (0.62,0.85)	-	0.82 (0.69,0.90)
S3a	Train	0.50 (0.38,0.62)	0.62 (0.50,0.73)	∼50M	0.70 (0.58,0.80)	∼100M	0.58 (0.46,0.69)	∼50M	0.84 (0.74,0.91)
	Test	0.61 (0.47,0.74)	0.59 (0.45,0.72)	-	0.57 (0.43,0.70)	-	0.63 (0.49,0.75)	-	0.73 (0.60,0.84)
Rp	Train	0.75 (0.63,0.84)	0.88 (0.77,0.94)	∼50M	0.89 (0.79,0.95)	∼100M	0.88 (0.77,0.94)	∼30M	0.75 (0.63,0.84)
	Test	0.84 (0.71,0.91)	0.94 (0.83,0.98)	-	0.88 (0.76,0.94)	-	0.88 (0.76,0.94)	-	0.84 (0.71,0.91)

TABLE XIV: Accuracies/Proportions with 95% Wilson confidence intervals and approximate token cost for training (rounded to the upper 10M) for different versions of ProTeGi (2, 5 iterations) and Prompt Enrichment (PE) concept across target variables (S1, S3a, Rp) and datasets (Train, Test). Starting and Manual prompts are shown without costs.

underlying prompt space by focusing only on promising subspace parts. However, the results must be taken with caution, considering the relatively small datasets and the observed persistence problem in LLMs.

V. Discussion

A. Answering research questions

This study found that automated prompt optimization can directly benefit information extraction workflows at ING Bank, particularly the task of extracting certain variables from long sustainability reports.

1) Automated prompt optimization showed effectiveness on the extraction of certain target variables by increasing accuracy over the initial prompts: On the test set, automated prompt optimization improved the accuracy of the starting prompts across all three targets. S1 increased from **53%** to **80%** (by **27pp**); S3a from **61%** to **65%** (by **4pp**); Rp from **84%** to **94%** (by **10pp**) (see Table VII). After the Holm-Bonferroni correction, only S1 remains statistically significant ($p = 0.0005$); Rp is borderline but not significant; S3a is not significant. This reflects the smaller effect sizes and the limited power of the study due to the small data sets.

Beyond accuracy, the optimized prompts also show higher stability on two of the three target variables: response impersistence drops on the test set from **22%→8%** (S1) and **8%→0%** (Rp); rises **6%→12%** for S3a. The impersistence of the evaluation shows a similar trend. See Tables X, XI and Figures 3, 4, 5 for more information.

2) Automated prompt optimization showed similar performance to expert-managed manual prompt engineering on some variables: Across targets, point estimates varied, telling a mixed story. On S1, the manual prompt showed slightly higher performance (Optimized: **80%** vs. Manual: **82%**); on Rp, auto-optimized prompt was higher by **10pp** (Optimized: **94%** vs. Manual: **84%**); On S3a, manual was higher by **8pp** (Optimized: **65%** vs. Manual: **73%**) (See Table VIII). The corresponding 95% confidence intervals overlapped in each case.

Automated optimization produced competitive prompts relative to manual prompt engineering on S1 and Rp, but was outperformed on S3a. This likely highlights the task complexity of S3a: qualitatively, the auto-optimized prompt was a paraphrase of the initial prompt; in contrast, the manual prompt enumerated a list of detailed step-by-step instructions (Appendix). Consequently, this suggests that LLM feedback-based optimization might deliver smaller and less consistent improvements on more complex targets.

To formally compare paired predictions, the study applied the exact version of the two-sided McNemar test (binomial) to discordant pairs per target. All p-values exceeded 0.05 and remained insignificant after Holm-Bonferroni correction. However, these tests were underpowered and produced imprecise estimates given the small number of discordant pairs (3 for S1, 16 for S3a, 5 for Rp). The author did not conduct formal equivalence testing; this would require prespecified margins and additional data.

In general, the methods demonstrated comparable point estimate performance, with fluctuations in either direction depending on the target variable. The study did not find enough evidence to reject the null and claim that the methods showed a significant difference.

In practice, this suggests that automated optimization can produce prompts competitive in performance to manually engineered ones. This effectively enables more scalable LLM-based automation solutions for enterprise scenarios, reducing the manual effort needed for prompt engineering. However, the effect is not universal; manual prompts outperformed auto-optimized on one variable, while losing on another.

B. Task dependence and iteration budget

The task's difficulty closely moderates the gains observed. For relatively straightforward numeric extraction of the S1 target variable, five iterations of the ProTeGi algorithm already captured most of the improvement; the ten iteration version of the algorithm only added

Var	Data	Starting	ProTeGi-5it		ProTeGi-10it		ProTeGi-5it + GradV		Manual
			Acc.	Tokens	Acc.	Tokens	Acc.	Tokens	
S1	Train	0.47 (0.35,0.59)	0.78 (0.67,0.86)	∼100M	0.75 (0.63,0.84)	∼200M	0.70 (0.58,0.80)	∼100M	0.70 (0.58,0.80)
	Test	0.53 (0.39,0.66)	0.76 (0.62,0.85)	-	0.80 (0.66,0.89)	-	0.80 (0.66,0.89)	-	0.82 (0.69,0.90)
S3a	Train	0.50 (0.38,0.62)	0.70 (0.58,0.80)	∼100M	0.72 (0.60,0.81)	∼200M	0.70 (0.58,0.80)	∼90M	0.84 (0.74,0.91)
	Test	0.61 (0.47,0.74)	0.57 (0.43,0.70)	-	0.65 (0.51,0.77)	-	0.71 (0.58,0.82)	-	0.73 (0.60,0.84)
Rp	Train	0.75 (0.63,0.84)	0.89 (0.79,0.95)	∼100M	0.84 (0.74,0.91)	∼200M	0.89 (0.79,0.95)	∼70M	0.75 (0.63,0.84)
	Test	0.84 (0.71,0.91)	0.88 (0.76,0.94)	-	0.94 (0.83,0.98)	-	0.92 (0.81,0.97)	-	0.84 (0.71,0.91)

TABLE XV: Accuracies/Proportions with 95% Wilson confidence intervals and approximate token cost for training (rounded to the upper 10M) for different versions of ProTeGi (2, 5 iterations) and ProTeGi + GradV (5 iterations) across target variables (S1, S3a, Rp) and datasets (Train, Test). Starting and Manual prompts are shown without costs.

4pp to the test set accuracy (from **76%→80%**) (Table XIII). In contrast, the more context-reliant S3a saw any improvements only at full ten iterations, unless a verification gate is added (see the following sections). Rp has a high starting accuracy at **84%** and quickly peaks at **94%** even after two iteration versions.

Practically, this suggests a need for a dynamic prompt optimization heuristic. The study derives a hypothesis for an example heuristic from narrow tasks (S1, S3a, Rp) on the dataset at hand: begin with five iterations for simple fields; if the task is more complex and context-dependent, either increase to ten, or experiment with adding gradient verification; if the task is not improved even after 10 iterations or GradV modification, manual prompt engineering might be considered. In such cases, the proposed prompt enrichment (PE) algorithm could be a valuable tool to facilitate manual prompt engineering (see the following sections). See Tables XIII, XV, and related result sections for a more detailed description.

C. Stability effect in production environment

In addition to accuracy, the study observed and documented substantial variability between identical inference runs, even at low temperature. In addition, a considerable difference was observed between the variability of Gemini Flash 2.0 and Gemini Flash-Lite 2.0 (see Tables VI, V). Optimized prompts reduce this variability on S1 and Rp compared to the starting prompts, thereby lowering the corresponding likelihood of random label flips across identical inference runs. In contrast, the variability increased on S3a for the optimized prompt. The reported stability gain or loss is often omitted in optimization papers, yet it is relevant for large-scale enterprise extraction and evaluation pipelines. This also highlights the inherent randomness of LLMs, where even under controlled stochasticity (near-zero temperature of 0.01), the label flips can be common, emphasizing evaluation challenges for the growing field of practical LLM applications.

More broadly, the results of the stability experiment suggest that the model's deterministic behavior also depends on the quality of the prompts. Whether the observed effect is caused by higher accuracy or remains a separate variable entirely is unclear and needs further research.

D. Gradient Verification (GradV): a proposed verifier for LLM-feedback guided prompt optimization

The study introduces the Gradient Verification (GradV) procedure, a concept for a simple yet powerful decision gate that discards LLM feedback/candidate prompts if they fail to address the very problem they were generated to solve (Algorithm 1). Conceptually, GradV complements existing LLM feedback-based prompt optimizers, such as ProTeGi [9], AutoHint [20]; and the critic / self-refine family of LLM output improvement [38], [40].

Empirically, ProTeGi-5it with GradV outperforms standard ProTeGi-10it on the most challenging target variable of S3a and nearly achieves the performance of the manually engineered prompt on the test dataset. Simultaneously, a training cost reduction of ∼0–30% compared to standard ProTeGi-5it (or ∼50% compared to ProTeGi-10it) is present. Across the other two variables of S1 and Rp, ProTeGi-5it with GradV slightly beats ProTeGi-5it (and also matches ProTeGi-10it) at a comparable or lower cost, indicating that Gradient Verification can effectively guide the exploration process without sacrificing performance. For more information, see Table XV and the corresponding paragraphs.

Unlike self-refine/critic loops, the author positions GradV as a decision gate: LLM feedback is retained only if it corrects the generation error on the source data point (or batch), thereby pruning the search space and reducing the evaluation load (on UCB evaluation in the ProTeGi case). In cases of long-document information extraction, this can be particularly valuable, as each evaluation is costly.

E. Prompt Enrichment as transparent, low-cost prompt engineering aid framework

The study also introduced the prompt enrichment algorithm as a proof-of-concept method that converts vali-

dated LLM-generated error rationales into instructions directly appended to the prompt (Algorithm 2). The technique produces small but consistent gains over the initial prompt at a cost comparable to ProTeGi-2it and approximately half of ProTeGi-5it, while matching the performance of ProTeGi-5it on the test set (Table XIV).

Although not always competitive with the full ProTeGi-10it version, the principal value of the proposed algorithm lies in its transparency: it provides clear, human-readable rationales (explanations) of failure reasons that can serve as a seed for downstream prompt optimization or manual prompt engineering. More broadly, the algorithm provides a clear textual description of inference errors observed on the dataset, effectively forming dataset rationale error descriptions that can be used for other purposes beyond prompt optimization. The reasons generated for the three target variables are available in the Appendix.

In particular, the score of the final combined prompt on the train set is sometimes lower than that of the optimized individual components, indicating that there might be deeper connections between the elements and emphasizing the need for future research (see the Appendix).

F. Costs and the role of retrieval systems

The training costs observed in the study are dominated by costs for the inference of the entire document at $\sim$190M tokens per target, plus $\sim$3-7M tokens for feedback (see Table IX). A prior retrieval setup that narrows each report to relevant $\sim$10 pages would reduce the token budget by roughly 15× without altering the optimizer algorithm. Given the observed iteration-sensitivity heuristic, pairing a retrieval system with five-iteration optimization (and GradV if needed) could likely provide an optimal cost-performance trade-off.

G. Validity threats

The enforced output schema and the observed model persistence problem could confound the observed performance gains. Small data sets resulting from the specificity of the use case yield few discordant pairs and wide confidence intervals for proportions (accuracy), resulting in underpowered findings and preventing the testing of the equivalence hypothesis between different prompt optimization methods. In general, the study evaluates one model family on one particular domain and type of reports; generalization to other models and document types requires further study.

A special note must be made regarding the stability results: The study analyzes persistence at near-zero temperature, keeping the default top p and top k values, as they are applied before temperature sampling in Gemini [54]. This can confound the generalization of stability tests for theoretical configurations with increased determinism. The impact of setting a single predefined seed in the Gemini API is not explored, as the feature is in the preview phase and does not guarantee determinism [55]. Furthermore, Gemini black-box enforced schemas and residual non-determinism at near-zero temperature might interact, affecting both accuracy and stability; The study attempts to mitigate this by repeating runs and reporting EIP/RIP metrics.

Despite the configuration confounders, the study clearly demonstrates that, under certain conditions and tasks, determinism can also depend on the quality of the prompt.

Given the small test set (n = 49) and the task's heterogeneity, the effect estimates may be imprecise. Therefore, hypothesis tests are used and treated as supporting evidence rather than as definitive proof. Generalization to other models, corpora, and extraction targets requires larger controlled studies.

Finally, as the ProTeGi version is customized to fit the task better, adhering to constraints, the deployed algorithm in the study may not be the most efficient configuration. Instead, it represents an attempt to address the real-life limitations of the use case while still optimizing prompts effectively. Other configurations may yield better or worse results.

H. Connection to prior work

The results of the study align with similar research on automated prompt optimization (ProTeGi [9], AutoHint [20], and OPRO [18], among others), showing comparable steady gains with a task-dependent magnitude.

1) Addition to the literature: The author presents a field evaluation of automated prompt optimization in a small-data enterprise setting, where the process is constrained by long, heterogeneous inputs and limited ground truth availability, which is underrepresented in prior studies. In addition, the study quantifies and highlights stability as a key metric often overlooked in prompt optimization, while reporting somewhat consistent reductions in impersistence for optimized prompts.

Based on observations and experiments with prompt optimization, the author further introduces the GradV procedure, a verifier of LLM feedback that, when combined with ProTeGi, improves the hard S3a target while reducing training costs.

Finally, the study presents a proof-of-concept of a prompt enrichment algorithm, built solely on the LLM feedback mechanism with GradV. The algorithm is fundamentally different from the standard approach of iterative optimization, focusing on capturing transparent

and meaningful error rationales rather than optimizing prompts. The captured rationales enable human involvement in the optimization process, allowing for a better understanding of the underlying LLM behavior.

I. Implications of findings for practice (ING and similar enterprise information extraction settings)

The study demonstrates that for simpler extraction target variables (such as S1), ProTeGi-5it can be used by default. For more complex and context-heavy tasks (similar to S3a), a full-scale ProTeGi-10it or ProTeGi-5it with GradV could prove valuable. A prior retrieval setup is also recommended, shrinking the document sizes before optimization, which results in a training cost that is lower by a magnitude while preserving the design of the optimization workflow. Given the observation that stability also changes based on prompts, auto-optimized prompts might still be preferable to starting prompts if they improve stability, even when the improvement in accuracy is modest.

The author further demonstrates that automated prompt optimization is not always guaranteed to be universally successful; some targets (such as S3a) might still require manual prompt engineering, which marks transparency as a valuable element of prompt optimization. The study presents a proof of concept for such a method (PE), which could facilitate subsequent manual prompt engineering with clear and transparent descriptions of error rationales.

J. Future work

Future work should establish equivalence/non-inferiority margins while utilizing substantially larger datasets to quantify comparisons with manually engineered prompts and facilitate more effective in-between optimization method comparisons. This would also enable a further exploration of the proposed GradV extension.

Secondly, the impact of prior retrieval on GradV-enhanced ProTeGi for long-document tasks should be explored, together with the possibility of dynamic convergence criteria.

Third, future work should further investigate the proposed Prompt Enrichment algorithm, attempting to improve performance while benefiting from transparency (not often mentioned in the prompt optimization literature).

Finally, DSPy-style integration of the discussed prompt optimization algorithms should be explored for unified optimization, verification, and production-ready deployment.

VI. Conclusion

This paper has demonstrated that automated prompt optimization can be effectively applied to information extraction tasks with long contexts, yielding promising results even with minimal data for both optimization and analysis.

The improvement in auto-optimized prompts over the initial was clearly observed, with some target variables displaying statistically significant results confirmed with hypothesis testing. Furthermore, the auto-optimized prompts displayed competitive performance with manual prompt engineering on some target variables; however, the observed differences were minor, which, combined with the low dataset size, lowers the precision of the comparison and generalization. The study did not perform formal equivalence or non-inferiority testing, as the available test set is underpowered for this purpose.

In summary, prompt optimization can be an effective strategy for improving starting prompts in information extraction scenarios with limited data sets for both training and testing. No significant evidence was found for a substantial difference in performance between auto-optimized and manually engineered prompts on some variables in this study, suggesting that even in such constrained scenarios, automated prompt optimization can be considered a competitive alternative to manual prompt engineering.

Practically, for information extraction from long documents under small labeled datasets, ProTeGi-5it extended with GradV for more challenging targets (e.g. S3a) could offer a decent cost-accuracy tradeoff; if cost is essential, proposed Prompt Enrichment could provide quick transparent gains at roughly half the token budget of ProTeGi-5it; if automated prompt optimization fails, manual prompt engineering aided with transparent Prompt Enrichment may be considered.

VII. Responsible AI disclosure

This work utilizes Generative Artificial Intelligence. The author applied it for writing assistance, proofreading, and formatting LaTeX tables (from Pythonic results). Each bit of the formatted content was manually verified. Furthermore, AI was utilized as a coding assistant, identifying simple bugs and exploring various solutions.

Models used: Overleaf Writefull assistant, Grammarly writing assistant, GPT family, Gemini family, Microsoft Copilot.

Acknowledgements

I want to express my gratitude to my supervisors for their guidance, support, and valuable insights throughout this study.

The author thanks Dr. Jörg Osterrieder (University of Twente, BMS), Dr. Wallace Corbo Ugulino (University of Twente, EEMCS), Dr. Hazal Koptagel (ING), and MSc. Morris Groot Beumer (ING) for their time, expertise, supervision, and encouragement during the course of this research.

REFERENCES

[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. ukasz Kaiser, and I. Polosukhin, "Attention is All you Need," in *Advances in Neural Information Processing Systems*, vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html

[2] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language Models are Few-Shot Learners," Jul. 2020. [Online]. Available: http://arxiv.org/abs/2005.14165

[3] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, and G. Neubig, "Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing," *ACM Comput. Surv.*, vol. 55, no. 9, pp. 195:1–195:35, Jan. 2023. [Online]. Available: https://dl.acm.org/doi/10.1145/3560815

[4] J. Maharjan, A. Garikipati, N. P. Singh, L. Cyrus, M. Sharma, M. Ciobanu, G. Barnes, R. Thapa, Q. Mao, and R. Das, "OpenMedLM: Prompt engineering can out-perform fine-tuning in medical question-answering with open-source large language models," *Scientific Reports*, vol. 14, no. 1, p. 14156, Jun. 2024. [Online]. Available: https://www.nature.com/articles/s41598-024-64827-6

[5] H. Nori, Y. T. Lee, S. Zhang, D. Carignan, R. Edgar, N. Fusi, N. King, J. Larson, Y. Li, W. Liu, R. Luo, S. M. McKinney, R. O. Ness, H. Poon, T. Qin, N. Usuyama, C. White, and E. Horvitz, "Can Generalist Foundation Models Outcompete Special-Purpose Tuning? Case Study in Medicine," Nov. 2023. [Online]. Available: http://arxiv.org/abs/2311.16452

[6] A. Salinas and F. Morstatter, "The Butterfly Effect of Altering Prompts: How Small Changes and Jailbreaks Affect Large Language Model Performance," in *Findings of the Association for Computational Linguistics: ACL 2024*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 4629–4651. [Online]. Available: https://aclanthology.org/2024.findings-acl.275/

[7] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, and Y. Iwasawa, "Large Language Models are Zero-Shot Reasoners," Jan. 2023. [Online]. Available: http://arxiv.org/abs/2205.11916

[8] D.-K. Kim, S. Sohn, L. Logeswaran, D. Shim, and H. Lee, "MultiPrompter: Cooperative Prompt Optimization with Multi-Agent Reinforcement Learning," Oct. 2023. [Online]. Available: http://arxiv.org/abs/2310.16730

[9] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, and M. Zeng, "Automatic Prompt Optimization with "Gradient Descent" and Beam Search," Oct. 2023. [Online]. Available: http://arxiv.org/abs/2305.03495

[10] Y. Mao, J. He, and C. Chen, "From Prompts to Templates: A Systematic Prompt Template Analysis for Real-world LLMapps," Apr. 2025. [Online]. Available: http://arxiv.org/abs/2504.02052

[11] C. Olea, H. Tucker, J. Phelan, C. Pattison, S. Zhang, M. Lieb, D. Schmidt, and J. White, "Evaluating Persona Prompting for Question Answering Tasks," in *Security, Privacy and Trust Management*. Academy & Industry Research Collaboration Center, Jun. 2024, pp. 63–81. [Online]. Available: https://aircconline.com/csit/papers/vol14/csit141106.pdf

[12] G. Li, H. A. A. K. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem, "CAMEL: Communicative Agents for "Mind" Exploration of Large Language Model Society," Nov. 2023. [Online]. Available: http://arxiv.org/abs/2303.17760

[13] M. Zheng, J. Pei, L. Logeswaran, M. Lee, and D. Jurgens, "When "A Helpful Assistant" Is Not Really Helpful: Personas in System Prompts Do Not Improve Performances of Large Language Models," Oct. 2024. [Online]. Available: http://arxiv.org/abs/2311.10054

[14] S. Gupta, V. Shrivastava, A. Deshpande, A. Kalyan, P. Clark, A. Sabharwal, and T. Khot, "Bias Runs Deep: Implicit Reasoning Biases in Persona-Assigned LLMs," Jan. 2024. [Online]. Available: http://arxiv.org/abs/2311.04892

[15] J. Kim, N. Yang, and K. Jung, "Persona is a Double-edged Sword: Mitigating the Negative Impact of Role-playing Prompts in Zero-shot Reasoning Tasks," Oct. 2024. [Online]. Available: http://arxiv.org/abs/2408.08631

[16] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba, "Large Language Models Are Human-Level Prompt Engineers," Mar. 2023. [Online]. Available: http://arxiv.org/abs/2211.01910

[17] T. Zhang, X. Wang, D. Zhou, D. Schuurmans, and J. E. Gonzalez, "TEMPERA: Test-Time Prompting via Reinforcement Learning," Nov. 2022. [Online]. Available: http://arxiv.org/abs/2211.11890

[18] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen, "Large Language Models as Optimizers," Apr. 2024. [Online]. Available: http://arxiv.org/abs/2309.03409

[19] A. Prasad, P. Hase, X. Zhou, and M. Bansal, "GrIPS: Gradient-free, Edit-based Instruction Search for Prompting Large Language Models," in *Proceedings of the 17th Conference of the European Chapter of the Association for Computational Linguistics*, A. Vlachos and I. Augenstein, Eds. Dubrovnik, Croatia: Association for Computational Linguistics, May 2023, pp. 3845–3864. [Online]. Available: https://aclanthology.org/2023.eacl-main.277/

[20] H. Sun, X. Li, Y. Xu, Y. Homma, Q. Cao, M. Wu, J. Jiao, and D. Charles, "AutoHint: Automatic Prompt Optimization with Hint Generation," Aug. 2023. [Online]. Available: http://arxiv.org/abs/2307.07415

[21] S. Yang, Y. Wu, Y. Gao, Z. Zhou, B. B. Zhu, X. Sun, J.-G. Lou, Z. Ding, A. Hu, Y. Fang, Y. Li, J. Chen, and L. Yang, "AMPO: Automatic Multi-Branched Prompt Optimization," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 20 267–20 279. [Online]. Available: https://aclanthology.org/2024.emnlp-main.1130/

[22] Y. Chen, J. Arkin, Y. Hao, Y. Zhang, N. Roy, and C. Fan, "PRompt Optimization in Multi-Step Tasks (PROMST): Integrating Human Feedback and Heuristic-based Sampling," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds. Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 3859–3920. [Online]. Available: https://aclanthology.org/2024.emnlp-main.226/

[23] E. Levi, E. Brosh, and M. Friedmann, "Intent-based Prompt Calibration: Enhancing prompt optimization with synthetic boundary cases," Feb. 2024. [Online]. Available: http://arxiv.org/abs/2402.03099

[24] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan, "Tree of Thoughts: Deliberate Problem Solving with Large Language Models," Dec. 2023. [Online]. Available: http://arxiv.org/abs/2305.10601

[25] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, "Self-Consistency Improves Chain of Thought Reasoning in Language Models," Mar. 2023. [Online]. Available: http://arxiv.org/abs/2203.11171

[26] Z. Zhang, A. Zhang, M. Li, and A. Smola, "Automatic Chain of Thought Prompting in Large Language Models," Oct. 2022. [Online]. Available: http://arxiv.org/abs/2210.03493

[27] L. Wang, N. Yang, and F. Wei, "Learning to Retrieve In-Context Examples for Large Language Models," Jan. 2024. [Online]. Available: http://arxiv.org/abs/2307.07164

[28] C. Qin, A. Zhang, C. Chen, A. Dagar, and W. Ye, "In-Context Learning with Iterative Demonstration Selection," Dec. 2024. [Online]. Available: http://arxiv.org/abs/2310.09881

[29] E. Agarwal, J. Singh, V. Dani, R. Magazine, T. Ganu, and A. Nambi, "PromptWizard: Task-Aware Prompt Optimization Framework," Oct. 2024. [Online]. Available: http://arxiv.org/abs/2405.18369

[30] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang, "Connecting Large Language Models with Evolutionary Algorithms Yields Powerful Prompt Optimizers," Feb. 2024. [Online]. Available: http://arxiv.org/abs/2309.08532

[31] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel, "Promptbreeder: Self-Referential Self-Improvement Via Prompt Evolution," Sep. 2023. [Online]. Available: http://arxiv.org/abs/2309.16797

[32] W. Cui, J. Zhang, Z. Li, H. Sun, D. Lopez, K. Das, B. Malin, and S. Kumar, "PhaseEvo: Towards Unified In-Context Prompt Optimization for Large Language Models," Feb. 2024. [Online]. Available: http://arxiv.org/abs/2402.11347

[33] M. Deng, J. Wang, C.-P. Hsieh, Y. Wang, H. Guo, T. Shu, M. Song, E. P. Xing, and Z. Hu, "RLPrompt: Optimizing Discrete Text Prompts with Reinforcement Learning," Oct. 2022. [Online]. Available: http://arxiv.org/abs/2205.12548

[34] H. Xu, Y. Chen, Y. Du, N. Shao, Y. Wang, H. Li, and Z. Yang, "GPS: Genetic Prompt Search for Efficient Few-shot Learning," Oct. 2022. [Online]. Available: http://arxiv.org/abs/2210.17041

[35] J. Cheng, X. Liu, K. Zheng, P. Ke, H. Wang, Y. Dong, J. Tang, and M. Huang, "Black-Box Prompt Optimization: Aligning Large Language Models without Model Training," Jun. 2024. [Online]. Available: http://arxiv.org/abs/2311.04155

[36] C. Yeung, J. Yu, K. C. Cheung, T. W. Wong, C. M. Chan, K. C. Wong, and K. Fujii, "A Zero-Shot LLM Framework for Automatic Assignment Grading in Higher Education," Jan. 2025. [Online]. Available: http://arxiv.org/abs/2501.14305

[37] H. Seo, T. Hwang, J. Jung, H. Kang, H. Namgoong, Y. Lee, and S. Jung, "Large Language Models as Evaluators in Education: Verification of Feedback Consistency and Accuracy," *Applied Sciences*, vol. 15, no. 2, p. 671, Jan. 2025. [Online]. Available: https://www.mdpi.com/2076-3417/15/2/671

[38] N. McAleese, R. M. Pokorny, J. F. C. Uribe, E. Nitishinskaya, M. Trebacz, and J. Leike, "LLM Critics Help Catch LLM Bugs," Jun. 2024. [Online]. Available: http://arxiv.org/abs/2407.00215

[39] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao, "Reflexion: Language Agents with Verbal Reinforcement Learning," Oct. 2023. [Online]. Available: http://arxiv.org/abs/2303.11366

[40] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark, "Self-Refine: Iterative Refinement with Self-Feedback," May 2023. [Online]. Available: http://arxiv.org/abs/2303.17651

[41] J. Li, J. Chen, R. Ren, X. Cheng, X. Zhao, J.-Y. Nie, and J.-R. Wen, "The Dawn After the Dark: An Empirical Study on Factuality Hallucination in Large Language Models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds. Bangkok, Thailand: Association for Computational Linguistics, Aug. 2024, pp. 10 879–10 899. [Online]. Available: https://aclanthology.org/2024.acl-long.586/

[42] G. Perković, A. Drobnjak, and I. Botički, "Hallucinations in LLMs: Understanding and Addressing Challenges," in *2024 47th MIPRO ICT and Electronics Convention (MIPRO)*, May 2024, pp. 2084–2088. [Online]. Available: https://ieeexplore.ieee.org/document/10569238

[43] M. Chelli, J. Descamps, V. Lavoué, C. Trojani, M. Azar, M. Deckert, J.-L. Raynier, G. Clowez, P. Boileau, and C. Ruetsch-Chelli, "Hallucination Rates and Reference Accuracy of ChatGPT and Bard for Systematic Reviews: Comparative Analysis," *Journal of Medical Internet Research*, vol. 26, no. 1, p. e53164, May 2024. [Online]. Available: https://www.jmir.org/2024/1/e53164

[44] Z. Xu, S. Jain, and M. Kankanhalli, "Hallucination is Inevitable: An Innate Limitation of Large Language Models," Feb. 2025. [Online]. Available: http://arxiv.org/abs/2401.11817

[45] Y. Chen, X. Pan, Y. Li, B. Ding, and J. Zhou, "Simple and Provable Scaling Laws for the Test-Time Compute of Large Language Models," May 2025. [Online]. Available: http://arxiv.org/abs/2411.19477

[46] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba, "Evaluating Large Language Models Trained on Code," Jul. 2021. [Online]. Available: http://arxiv.org/abs/2107.03374

[47] O. Khattab, A. Singhvi, P. Maheshwari, Z. Zhang, K. Santhanam, S. Vardhamanan, S. Haq, A. Sharma, T. T. Joshi, H. Moazam, H. Miller, M. Zaharia, and C. Potts, "DSPy: Compiling Declarative Language Model Calls into Self-Improving Pipelines," Oct. 2023. [Online]. Available: http://arxiv.org/abs/2310.03714

[48] "Stanfordnlp/dspy," Stanford NLP, Aug. 2025. [Online]. Available: https://github.com/stanfordnlp/dspy

[49] "Optimizers - DSPy." [Online]. Available: https://dspy.ai/learn/optimization/optimizers/

[50] K. Ramnath, K. Zhou, S. Guan, S. S. Mishra, X. Qi, Z. Shen, S. Wang, S. Woo, S. Jeoung, Y. Wang, H. Wang, H. Ding, Y. Lu, Z. Xu, Y. Zhou, B. Srinivasan, Q. Yan, Y. Chen, H. Ding, P. Xu, and L. L. Cheong, "A Systematic Survey of Automatic Prompt Optimization Techniques," Feb. 2025. [Online]. Available: http://arxiv.org/abs/2502.16923

[51] "Document understanding | Gemini API." [Online]. Available: https://ai.google.dev/gemini-api/docs/document-processing

[52] P. Auer, N. Cesa-Bianchi, and P. Fischer, "Finite-time Analysis of the Multiarmed Bandit Problem," *Machine Learning*, vol. 47, no. 2, pp. 235–256, May 2002. [Online]. Available: https://doi.org/10.1023/A:1013689704352

[53] V. Kuleshov and D. Precup, "Algorithms for multi-armed bandit problems," Feb. 2014. [Online]. Available: http://arxiv.org/abs/1402.6028

[54] "Prompt design strategies | Gemini API." [Online]. Available: https://ai.google.dev/gemini-api/docs/prompting-strategies

[55] "Experiment with parameter values | Generative AI on Vertex AI." [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs/learn/prompts/adjust-parameter-values

[56] "Image understanding | Generative AI on Vertex AI." [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs/multimodal/image-understanding

[57] E. B. Wilson, "Probable Inference, the Law of Succession, and Statistical Inference," *Journal of the American Statistical Association*, vol. 22, no. 158, pp. 209–212, 1927. [Online]. Available: https://www.jstor.org/stable/2276774

[58] S. Wallis, "Binomial Confidence Intervals and Contingency Tests: Mathematical Fundamentals and the Evaluation of Alternative Methods," *Journal of Quantitative Linguistics*, vol. 20, no. 3, pp. 178–208, Aug. 2013. [Online]. Available: https://doi.org/10.1080/09296174.2013.799918

[59] A. Agresti, *An Introduction to Categorical Data Analysis*, third edition ed., ser. Wiley Series in Probability and Statistics. Hoboken, NJ: John Wiley & Sons, 2019.

[60] Q. McNemar, "Note on the sampling error of the difference between correlated proportions or percentages," *Psychometrika*, vol. 12, no. 2, pp. 153–157, Jun. 1947. [Online]. Available: https://doi.org/10.1007/BF02295996

[61] K. J. Nicholson, M. Sherman, S. N. Divi, D. R. Bowles, and A. R. Vaccaro, "The Role of Family-wise Error Rate in Determining Statistical Significance," *Clinical Spine Surgery*, vol. 35, no. 5, p. 222, Jun. 2022. [Online]. Available: https://journals.lww.com/jspinaldisorders/abstract/2022/06000/the_role_of_family_wise_error_rate_in_determining.7.aspx

[62] S. Holm, "A Simple Sequentially Rejective Multiple Test Procedure," *Scandinavian Journal of Statistics*, vol. 6, no. 2, pp. 65–70, 1979. [Online]. Available: https://www.jstor.org/stable/4615733

[63] "Gemini 2.0 Flash-Lite | Generative AI on Vertex AI | Google Cloud." [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash-lite

[64] "Gemini 2.0 Flash | Generative AI on Vertex AI." [Online]. Available: https://cloud.google.com/vertex-ai/generative-ai/docs/models/gemini/2-0-flash

APPENDIX

A. Gemini hyperparameters

The hyperparameters for reproducibility are provided in Table XVI.

Hyperparameter	Flash	Flash-Lite
temperature	0.01 (1.0 for feedback)	0.01
top p	0.95 (1.0 for feedback)	0.95
top k (fixed by vendor for both models)	64	64
candidate count	1	1
max output tokens	8,192	8,192

TABLE XVI: Generation hyperparameters for Gemini 2.0 Flash and Flash-Lite used in this study. Flash-Lite is only used for inference and is limited to a single experiment. Only temperature is adjusted, the rest is set to defaults from the official documentation (except disabling top p filtering for training component) [63], [64]

B. ProTeGi Training Parameters

See Table XVII. The difference between runs is only in the number of training iterations.

Parameter	Explanation	Full	Downscaled
beam_width	Number of prompts in a beam	4	4
num_iterations	Number of training iterations	**10**	**2/5**
batch_size_for_errors	How many examples to sample when looking for errors	10	10
max_num_examples	Maximum number of examples used for one prompt to make gradients	1	1
num_gradients	Number of gradients generated for each prompt in the beam using incorrect examples	4	4
steps_per_gradient	Number of prompts generated per one gradient	1	1
num_variations	Number of paraphrases per one prompt	0	0
bandit_budget	Number of UCB evaluation iterations	10	10
bandit_sample_size	Number of data points used for evaluating each prompt evaluated in parallel during a single UCB iteration	10	10
c	UCB Exploration constant	1	1
num_prompts_per_round	Prompts evaluated in parallel by UCB	4	4
num_threads	Parallel threads	20	20

TABLE XVII: ProTeGi training parameters

C. ProTeGi Demo Example

See Table XVIII.

D. Starting and Manual prompts

See Table XIX.

E. Prompts optimized with ProTeGi

See Table XX. Note that the prompts in this Appendix are auto-generated artifacts of the prompt optimization pipeline and are included for transparency. Discrepancies, minor inconsistencies, and redundancies are expected and showcase how auto-optimization differs from manual engineering.

F. Prompt Enrichment error rationales

See Tables XXI, XXII, XXIII for a post-pooling prefiltering error rationales. Note that the accuracies are from the training set and were calculated during runtime. Hence, the starting accuracy might differ slightly from the one reported in the paper, due to persistence issues discussed. The reasons in this Appendix are auto-generated artifacts of the proposed prompt optimization pipeline and are

Algorithm component	Example
Starting prompt - 5% accuracy	*How many letters r are in the given word?*
Dataset	[cranberry, strawberry, blackberry, tomato, etc.]
Beam of size 1	[”How many letters r are in the given word?”]
Examples evaluated	[strawberry, cranberry]
Incorrect examples	cranberry: predicted 2; strawberry: predicted 2;
Textual gradient	The prompt, while seemingly straightforward, doesn't explicitly address the case sensitivity. It's possible the model is only counting lowercase 'r' and missing any uppercase 'R' present in the examples. Therefore, words like 'River' or 'Roger' would be miscounted.
New prompt candidate	Count the occurrences of the letter 'r', regardless of case, in the following word.
UCB evaluation	”How many letters r are in the given word”: ”reward”: 0.29, ”Count the occurrences of the letter 'r', regardless of case, in the following word.”: ”reward”: 0.16
New beam (replaces the old)	[”How many letters r are in the given word?”]
Final selection	No need, only one prompt in the beam.
Final prompt - 100% accuracy	*“Scan the word from left to right. Start with a count of zero. Each time you encounter the letter r, add one to your count. What is the final count?”*

TABLE XVIII: Demo example of customized ProTeGi algorithm used in this study. The example is produced by running the algorithm on a toy use case. The arrow shows the algorithm repeating itself for a number of iterations specified by a hyperparameter.

included for transparency. Discrepancies, minor inconsistencies, and redundancies are expected and showcase how auto-optimization differs from manual engineering.

See Tables XXIV, XXV, XXVI for post-filtering (non-contradicting, non-performance decreasing) rationales. They are included in the final prompt.

G. ProTeGi Meta prompts

For ProTeGi meta-prompts see Table XXVII.

H. Visualization of document evaluation changes under controlled experiment conditions

Figures 3, 4, 5 show evaluation persistence across 10 iterations for each document in the format of the model's ”certainty”. Certainty is defined as the metric that highlights the proportion of correct and incorrect classifications for each document in ten iterations. A certainty of one means that the document was correct or incorrect ten times. A certainty of zero means that the document was correct five times and incorrect another five.

I. Prompt Enrichment Meta prompts

See Table XXVIII.

Key	Starting Prompt	Manual Prompt	Schemas
S3a	Does the company have scope 3 assurance for the reporting year 2023?	Extract assurance information from the following file for the year 2023. **Instructions:** * First, check if the file mentions the word 'assurance' or not. If there are no mentions of 'assurance': set has_scope_3_assurance to null. * Second, if the file mentions the word 'assurance', check if an assurance was done by an independent professional assurance provider, such as an auditing firm. * If an assurance was done (limited or full): set has_scope_3_assurance.answer to True. * If an assurance was NOT done: set has_scope_3_assurance.answer to False. * If the company does not seek external assurance: set has_scope_3_assurance.answer to False. **Note:** * In order to have an assurance, the greenhouse gas (GHG) emission data must be subject to audit or review by an independent professional assurance provider, such as an auditing firm.	class Scope3EmissionsAssuranceSchema: has_scope_3_assurance: bool | None
S1	What is the Scope 1 greenhouse gas emission of this company in 2023?	Extract absolute scope 1 emissions from the following file for the year 2023. 10,000 tCO2 is 10000.0 tCO2e. 35,125 tCO2e is 35125.0 tCO2e. Pay attention to the unit and the multiplier of the value, convert it to tCO2e and return as a float. For example, if the absolute scope 1 emissions are 3.12 kilo tCO2e in the file, return 3120.0. If none is mentioned, return 0.0.	class AbsoluteScope1EmissionsSchema: total_scope1_emissions: float | None
Rp	Extract the start and end reporting date from the following file.	Extract the following fields from the following file: "start_date": Starting date of the reporting period of the report. Return in format YYYY-MM. If the report covers year 2020, the start date is 2020-01. If the report covers the year until end of April 2020, the start date is 2019-05. "end_date": Ending date of the reporting period of the report. Return in format YYYY-MM. If the report covers year 2020, the end date is 2020-12. If the report covers the year until end of April 2020, the end date is 2020-04.	class ReportingPeriodSchema start_date: str | None (YYYY-MM) desc: "Starting date of the reporting period of the report. Return in format YYYY-MM. Return None if not found." end_date: str | None (YYYY-MM) desc: "Ending date of the reporting period of the report. Return in format YYYY-MM. Return None if not found."

TABLE XIX: Starting and Manual prompts with corresponding schemas

Key	ProTeGi-2it	ProTeGi-5it	ProTeGi-5it + GradV	ProTeGi-10it
S1	Provide the Scope 1, Scope 2, and Scope 3 greenhouse gas emissions of the company in 2023, expressed in metric tons of carbon dioxide equivalent (MTCO2e). If combined emission values are available (e.g., Scope 1+2, Scope 2+3), also extract those, clearly indicating the scopes included in each value. What is the total amount of greenhouse gas emissions, expressed in metric tons of carbon dioxide equivalent (MT CO2e)?	Extract the *total* greenhouse gas emissions reported under Scope 1, encompassing *all* facilities and operations, for *the entire year 2023*. Provide the numerical value in *tonnes of CO2 equivalent (tCO2e)*. If the document fails to specify the total Scope 1 emissions for the whole year 2023 *in tCO2e*, or expresses it in different units, perform unit conversion based on the most reliable, up-to-date science. If no usable Scope 1 greenhouse gas emissions data at all can be identified for the whole of 2023, return 0.0. For example if the greenhouse gas emission is given as 0.15 MtCO2e transform it in 150000 tCO2e. Use the term total_scope1_emissions in the response. For example: total_scope1_emissions=150000. Also be sure to select only numerical values even if the string has thousand separators, ex. '1,200,000'. Return only the processed numerical value.,	Extract the Scope 1 greenhouse gas emissions for the year 2023. Ensure the output is in tonnes of CO2 equivalent (tCO2e). If the value is in KtCO2e, convert it to tCO2e.	Extract the company's ACTUAL Scope 1 greenhouse gas emissions for the year 2023. Prioritize a direct, explicitly stated value for TOTAL, ACTUAL Scope 1 emissions in 2023, reported in tons of CO2 equivalent (tCO2e). If a single, explicit value is unavailable, carefully calculate the TOTAL ACTUAL Scope 1 emissions for 2023 by summing all component ACTUAL emissions for 2023 ONLY. Ensure ALL values are converted to, and the final answer is expressed in, tons of CO2 equivalent (tCO2e). Critically, pay close attention to units, including prefixes like "k" (kilo - thousands), "M" (mega - millions), and "G" (giga - billions). Accurately convert any reported value to tons of CO2 equivalent. Ignore any targets, projections, or values from other years; focus SOLELY on actual 2023 emissions. Be precise, pay attention to numerical formatting. Output ONLY the final numerical value as a decimal number, without punctuation or units. If the information is not found or cannot be reliably converted, output 0.0.
S3a	Does this report mention any external assurance or verification related to its greenhouse gas (GHG) emissions data, including Scope 3 emissions? Provide a boolean indicating yes or no.	Does the document describe any independent assessment, verification, audit, or review of the company's performance related to sustainability (environmental impact, environmental management, etc.)? If so, detail the assessment. Include: the scope (what aspects are assessed), the nature of the assurance or validation (e.g., independent audit, adherence to standards), and the entities involved in conducting the assessment. If no such activity is described, indicate None.	Does the company's sustainability reporting for 2023 indicate any form of independent verification or external validation related to greenhouse gas emissions, including those related to its value chain or supply chain (even if not explicitly defined as 'Scope 3')? Respond True if any assurance activity is mentioned related to emissions beyond Scope 1 and 2; otherwise, False. If no information is present, the field should be None.	Does an independent external organization confirm the accuracy and reliability of the environmental, social, and governance (ESG) data reported for the year 2023? Answer "True" if an independent review, validation, or attestation has been performed. Answer "False" if an internal review, self-assessment, or no review has been performed. Answer "None" if this assessment is not applicable.
Rp	Extract the start and end dates of the ESG report from the following document. If the dates are not explicitly stated, infer them from the context of the report content. Date should be in YYYY-MM format.	Extract the reporting period's start and end dates from the following document, expressing the dates in 'YYYY-MM' format. If a specific day isn't mentioned, assume the last day of the month for end date and the first day for start date. If the reporting period can't be determined, or its format doesn't fit 'YYYY-MM', write None for both. Think broadly about the document content to infer all period information.	Extract the reporting period's start and end dates covered by this document. Prioritize dates explicitly stated. If an explicit date is unavailable, *infer* the reporting period based on context. The document likely summarizes financials, or key activities over a specific annual period. If the document uses phrases like 'as of [date]' heavily, consider that date as the end date and try to infer the annual period before it. Only return null if absolutely no date information, explicit, or inferable exists. Use YYYY-MM format.	Extract the central reporting timeframe from this document, formatted as YYYY-MM to YYYY-MM. Prioritize the 12-month period most explicitly linked to overall organizational financial results, performance metrics, or sustainability targets. Note a brief justification (1–2 sentences) for why this timeframe was chosen, referencing specific text within the document. While the document may contain descriptions of specific projects, focus on the date range that summarizes the organization's aggregate performance. Disregard date ranges related to solely future projections. If multiple potentially valid date ranges exist, prefer the 12-month range starting the closest to the latest March 31 date, but only choose it if it has actually occurred for that timeframe (prefer completed datasets). If no valid date can be found, output None, None with your reasons.

TABLE XX: ProTeGi-2it vs. ProTeGi-5it vs. ProTeGi-5it + GradV vs. ProTeGi-10it prompts.

Reason	Accuracy	Delta accuracy
Starting prompt: What is the Scope 1 greenhouse gas emission of this company in 2023?	0.53	0.00
If multiple greenhouse gas emissions are present, extract all values, and complete if that number is specified.	0.59	0.06
Extract the numerical value of 'Scope 1 emissions' in tCO2 eq. or related units, converting to metric tons (tonnes) if necessary, considering the specified unit (including conversions from kilotonnes by multiplying by 1000, '000 tonnes' to tonnes by multiplying by 1000, and MM MTCO2e by multiplying by 1000000), and outputting the exact number as a float, without rounding, including all digits; if no number is found, respond 'n/a'.	0.69	0.16
If the desired value is a range of values, determine its type. If it is an integer, select the closest one to the range.	0.59	0.06
When extracting numeric values, ignore any commas or periods inside the number.	0.64	0.11
If the exact year is not mentioned, identify data from the closest year.	0.47	-0.06

TABLE XXI: S1 prompt enrichment: reason, accuracy, and delta relative to the starting prompt. This is prior to contradiction filtering. Negative deltas are eliminated and not includes in the final prompt.

Reason	Accuracy	Delta accuracy
Starting prompt: Does the company have scope 3 assurance for the reporting year 2023?	0.50	0.00
If 'None' is not an option, default to 'False' if assurance information is absent or negative.	0.48	-0.02
When checking the existence of assurance, report true, even if it only relates to specific environmental components.	0.64	0.14
Use data tables containing percentages of suppliers committed to science-based targets to infer the existence of Scope 3 emissions accounting.	0.50	0.00
Determine Scope 3 assurance by considering mentions of 'external assurance,' 'assurance report,' 'independent assurance' of 'non-financial information,' 'third party' assurance, and explicit statements regarding Scope 3 emissions data, targets, or reporting; prioritize explicit mentions of 'assurance' or 'verification,' including any type of verification (limited review, independent review); consider approved programs or processes from international organizations; respond 'True' if explicit Scope 3 assurance is stated, 'None' if no explicit mention, and default to 'None' or null if the document doesn't explicitly mention Scope 3 assurance.	0.39	-0.11
Consider that 'financed emissions' can also constitute scope 3 emissions.	0.58	0.08
Prioritize sections dedicated to 'sustainability accounting' or 'climate accounting' when determining the presence scope 3 assurance.	0.53	0.03
Consider scope emissions reduction targets as evidence of commitment and possible assurance	0.47	-0.03

TABLE XXII: S3a prompt enrichment: reason, accuracy, and delta relative to the starting prompt. This is prior to contradiction filtering. Negative deltas are eliminated and not included in the final prompt.

Reason	Accuracy	Delta accuracy
Starting prompt: Extract the start and end reporting date from the following file.	0.84	0.00
Start date can be extracted in relationship with end date, for example: 'year from' or from a more precise month.	0.84	0.00
Focus on the current year's report, not historical data.	0.83	-0.01
When handling dates, especially start dates, consider the reporting period as a fiscal year from April to March.	0.78	-0.06
Only extract reporting dates if explicitly stated; do not assume any dates. If no explicit reporting dates are provided, return null for start and end dates.	0.73	-0.11
Return the start date with the assumption that the reporting starts on the first month of the year if no start month is specified.	0.88	0.04
Prioritize the '1 October 2022 - 30 September 2023' year to extract the reporting date.	0.77	-0.07
Confirm date ranges are for full reporting year.	0.86	0.02

TABLE XXIII: Rp prompt enrichment: reason, accuracy, and delta relative to the starting prompt. This is prior to contradiction filtering. Negative deltas are eliminated and not includes in the final prompt.

Reason	Accuracy	Delta accuracy
Starting prompt: What is the Scope 1 greenhouse gas emission of this company in 2023?	0.53	0.00
If multiple greenhouse gas emissions are present, extract all values, and complete if that number is specified.	0.59	0.06
Extract the numerical value of 'Scope 1 emissions' in tCO2 eq. or related units, converting to metric tons (tonnes) if necessary, considering the specified unit (including conversions from kilotonnes by multiplying by 1000, '000 tonnes' to tonnes by multiplying by 1000, and MM MTCO2e by multiplying by 1000000), and outputting the exact number as a float, without rounding, including all digits; if no number is found, respond 'n/a'.	0.69	0.16
When extracting numeric values, ignore any commas or periods inside the number.	0.64	0.11
Final prompt (combined)	**0.70**	**0.17**

TABLE XXIV: S1 prompt enrichment: reason, accuracy, and delta relative to the starting prompt after filtering. The final prompt is a concatenation of starting prompt and all reasons in the table.

Reason	Accuracy	Delta accuracy
Starting prompt: Does the company have scope 3 assurance for the reporting year 2023?	0.50	0.00
When checking the existence of assurance, report true, even if it only relates to specific environmental components.	0.64	0.14
Use data tables containing percentages of suppliers committed to science-based targets to infer the existence of Scope 3 emissions accounting.	0.50	0.00
Consider that 'financed emissions' can also constitute scope 3 emissions.	0.58	0.08
Prioritize sections dedicated to 'sustainability accounting' or 'climate accounting' when determining the presence scope 3 assurance.	0.53	0.03
Final prompt (combined)	**0.58**	**0.08**

TABLE XXV: S3a gradients: reason, accuracy, and delta relative to the starting prompt after filtering. The final prompt is a concatenation of starting prompt and all reasons in the table.

Reason	Accuracy	Delta accuracy
Starting prompt: Extract the start and end reporting date from the following file.	0.84	0.00
Return the start date with the assumption that the reporting starts on the first month of the year if no start month is specified.	0.88	0.04
Confirm date ranges are for full reporting year.	0.86	0.02
Final prompt (combined)	**0.88**	**0.04**

TABLE XXVI: Rp gradients: reason, accuracy, and delta relative to the starting prompt after filtering. The final prompt is a concatenation of starting prompt and all reasons in the table.

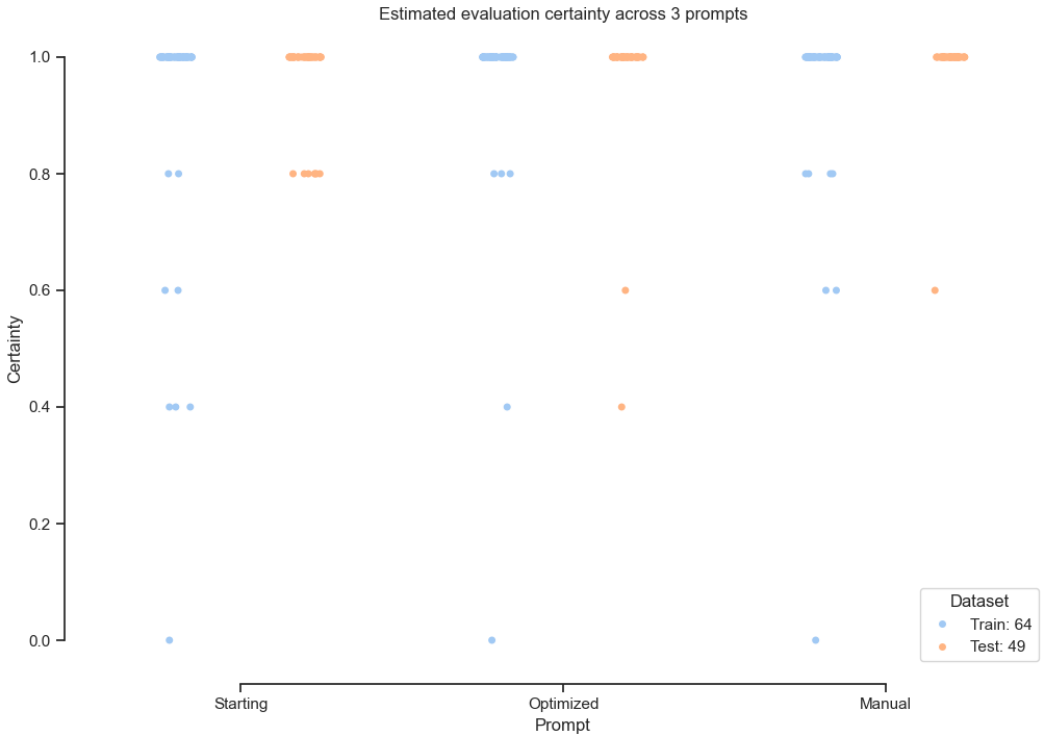

Fig. 3: Evaluation certainty for Scope 1 absolute emissions. The certainty score is 1 when the document has the same evaluation across all ten iterations and 0 when there is a 50/50 split.

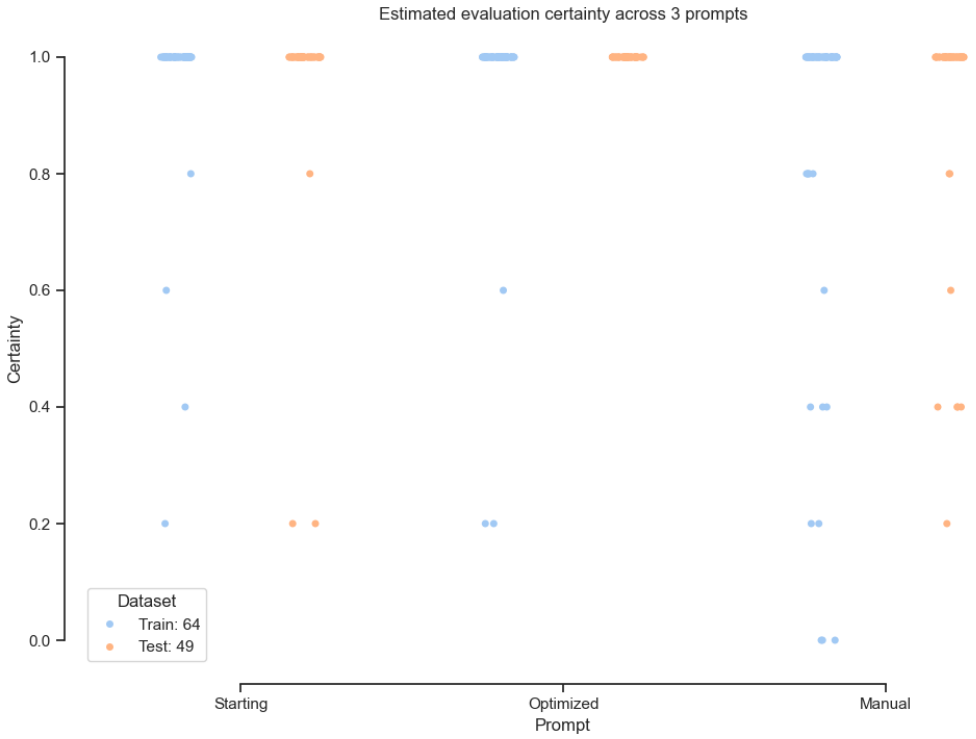

Fig. 4: Evaluation certainty for Reporting Period target. A score of 1 indicates full consistency across all iterations, while lower scores reflect mixed outcomes, with 0 indicating a perfectly mixed result of 5/5.

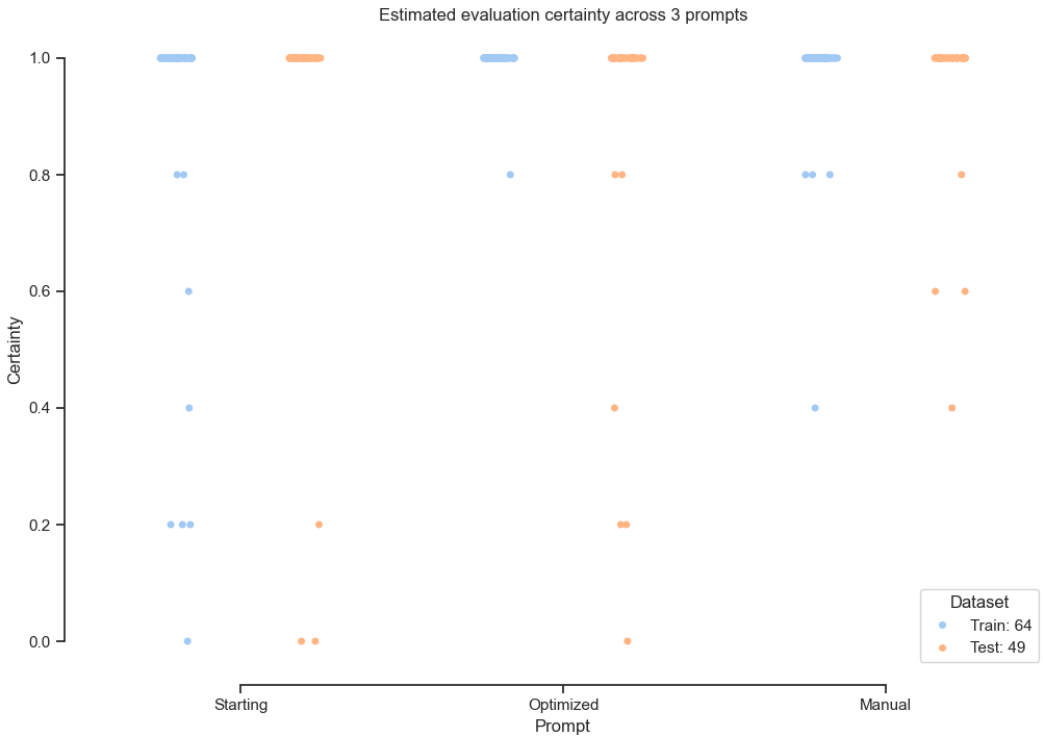

Fig. 5: Evaluation certainty for Scope 3 assurance. Certainty reflects the stability of classifications across ten iterations.

Prompt Name	Prompt Text
Generating gradients	I'm trying to write a zero-shot prompt for information extraction from pdfs. My current prompt is: "{prompt_text}" But this prompt gets the following example wrong: {incorrect_example} Give {num_gradients} reasons why the prompt could have gotten these examples wrong. Only give solid reasons, do not consider formatting of the result. **DO NOT GIVE MORE REASONS THAN {num_gradients}.** Wrap each reason with <START> and <END>. **WRAP EACH REASON WITH <START> and <END>, DO NOT MODIFY THE TAG.**
Generating new candidates	I'm trying to write a zero-shot prompt for information extraction from pdfs. My current prompt is: "{prompt_text}" But it gets the following examples wrong: {error_str} Based on these examples the problem with this prompt is that {gradient} Based on the above information, write {steps_per_gradient} different improved prompts. Wrap each prompt with <START> and <END>. **WRAP EACH PROMPT WITH <START> and <END>, DO NOT MODIFY THE TAG.** The {steps_per_gradient} new prompts are:

TABLE XXVII: Meta prompts for ProTeGi

Name	Prompt text
GRADIENT	I'm trying to write a zero-shot model for information extraction from pdfs. My current prompt is: "prompt_text" But with this prompt, the model gets following examples wrong: {error_str}. Analyze reasons for the mistake and list {num_gradients} fixes based on the prompt and the documents attached. Formulate each fix as a short instruction fixing the error, THAT WILL LATER BE APPENDED TO THE MAIN PROMPT. SO DIRECT THE INSTRUCTION TO THE MODEL THAT WILL PERFORM INFORMATION EXTRACTION (AS ADDITIONAL INSTRUCTIONS). Address the instruction to the model performing the task, not to the person asking this. Do not overfit to the document, the fix should be generalizable to other documents. Do not repeat the main prompt in your fixes, the fixes should be additive, not standalone. For context, some of the fixes will later be appended to the main prompt, so formulate them as short instructions additive to the prompt. Only give solid fixes, do not consider specific formatting of the result. DO NOT GIVE MORE FIXES THAN {num_gradients}. USE THE DOCUMENT I ATTACH TO THIS PROMPT FOR THE FEEDBACK, DO NOT GUESS. Use the schema attached to format your response (one variable in the schema = 1 fix). The {num_gradients} fixes are:
GROUP	I will give you a list of brief instructions for an information extraction LLM-backed system. The format of the instructions will be: Instruction 1: "text" Instruction 2: "text" and so on. The list will have some similar instructions, some different and some contradicting instructions. Your goal is to identify similar/identical instructions and combine similar/identical without touching different and contradicting. If the instructions are related and talk about same concept differently, combine them. Only return the combined instructions, unique, different and contradicting (do not return both new instructions combined from previous and also previous). Also combine prompts which talk about closely related stuff but in different words (and do not contradict). Identify similar/paraphrased/identical instructions in the given list and combine them. If the instructions are unique or have significant differences, or are contradicting, return them as is. If the instruction is "dirty", e.g. has some leftover tags in it, clean it up. Wrap each output instruction with <START> and <END>. WRAP EACH ONE WITH <START> and <END>, DO NOT MODIFY THE TAG. ONLY USE THE TAG FOR WRAPPING THE INSTRUCTIONS, Do not use it anywhere else.
CONTRADICTIONS	You are a heuristic filter for short task instructions. I will give you two instructions and you will identify whether they are directly contradicting (e.g. you cannot follow both at the same time). In other words, if following the first instruction makes it impossible to follow the second, or vice-versa, return True. If the instructions do not appear to contradict and can be executed together (by aggregating them), return False. ONLY RETURN TRUE IF THERE IS A REAL AND SERIOUS CONTRADICTION. The two instructions: {instruction_pair}
SIMILARITY	You are a binary similarity classifier between two short instructions. I will give you a pair of short instructions for an information extraction LLM-backed system. Identify whether instructions are similar/identical/related, or not. Instructions are considered similar if they talk about the same idea in different words and they do not contradict. The instructions are also similar if they are paraphrases of each other. Otherwise, they are not similar. Instruction 1: {instruction_1}; Instruction 2: {instruction_2}; Return True if similar, False if not. For context, the instructions are used as a part of a larger information extraction workflow, where I extract information from long documents using LLMs and a prompt. The prompt contains the starting part (basic task description, unchanged), while the short instructions are additives aimed at supplementing/improving/strengthening the starting prompt. Here is the starting prompt: {starting_prompt}. While analyzing similarity between the two instructions, keep this starting prompt in mind (to more effectively distinguish between similar and dissimilar based on the context knowledge, which is formulated as a starting prompt).
COMBINE	Here is a list of short instructions. Combine them into one (do not repeat similar stuff).

TABLE XXVIII: Meta prompts for prompt enrichment