Affective Prompt Engineering in Multi-Agent Systems

Blake Larkin

April 19, 2024

Abstract

Large language models notoriously feature certain undesirable characteristics which limit their problem-solving capabilities, such as the tendency for hallucination, stochastic outputs, and sensitivity to input variability. Recent model-external methods for addressing these properties of LLMs include Multi-Agent Systems (MAS) and prompt engineering (PE). However, there has yet to be research on combining verified PE methods with existing MASs, which could see easily realized gains in system output with minimal modification to system structure. This study explores this by applying affective PE to the ChatDev MAS on a pair of coding tasks of varying difficulty. The study found that contrary to what was hypothesized, affective PE in MASs on coding tasks was not associated with an increase in task performance, and was associated with increases in operational cost across all trials. This paper discusses the implications of these results, and recommends that designers carefully consider possible computational consequences of injecting affective PE into multi-agent systems, particularly with respect to unintended interactions between PE and general system behaviour.

1 Introduction

Large language models (LLMs) have seen large leaps in performance on key benchmarks in recent years, such as natural language processing, logic, mathematics, and programming tasks. Concurrently, research interests in these areas have also grown. While modifications to base models presents the most promising avenue of LLM development, a growing body of literature reports significant performance increases with model-external modifications to other aspects of LLM function and implementation (Han et al. 2024; Minaee et al. 2024; Sahoo et al. 2024). These modifications seek to maximize performance within a given model, thus circumventing the increase in cost associated with transitioning to a more advanced model (Bae et al. 2024). A large portion of these research efforts target key weaknesses of LLM performance such as hallucinations, unreliable performance in complex tasks, and sensitivity to variation in input. One of the earliest research efforts in this domain was the development of prompt engineering (PE), the process of designing task descriptions to improve model output. Building on initial discoveries of Zero-shot prompting (Radford et al. 2019) and Few-shot prompting (Brown et al. 2020), more complex methods later emerged such as Chain-of-Thought prompting (Wei et al. 2023), Tree-of-Thought prompting (Yao et al. 2023), and EmotionPrompt (Li et al. 2023). Further methods have been developed to leverage LLM-powered computational techniques, such as the Automatic Prompt Engineer (APE) (Zhou et al. 2023), or to model PE as an optimal control problem (Luo et al. 2023).

Current PE research has focused on the user-agent dyad, primarily in one-round interactions, with PE in multi-round interactions as a growing subset of current investigation. Research on PE in multi-agent systems is only beginning to emerge, due to the novelty of both fields (Luo et al. 2023). A multi-agent system (MAS) describes a set of cognitive agents, currently LLMs, which work collectively to perform some task. Similar to PE, these types of systems can mitigate some of the undesirable traits of LLMs without adjusting underlying model parameters. Collaborative agents can reduce hallucinations, (Hong et al. 2023), perform more complex and variable tasks compared to single-agents (Liu et al. 2023), and inherently reduce the stochastic nature of LLM output due to multi-round proofreading. MASs can be interpreted as the application of social decision-making theory to autonomous agents, with most extant MASs exhibiting characteristics of what has been dubbed a Reductionist approach of social decision-making: focusing on a collective group efficacy over an emergent coherence of agents seeking individual utility (Kalenka and Jennings 1999). The conversational structure of an MAS is generally a predetermined aspect of system behaviour, with current MASs displaying a wide range of inter-agent interaction schemas (Guo et al. 2024). Because of this predetermined structure, current PE techniques can be directly injected into systems' conversational behaviour, which could offer a cost-effective means to extend the performance gains of PE to MASs, without compromising their socially-mediated benefits.

To explore this novel dyamic, this study applied a verified affective PE method to a prebuilt software development MAS, and tested for change in system output along multiple variables, including code functionality, operational cost, and natural language metrics. The system was tested on two coding tasks of varying difficulty. Contrary to expectations, the study found that PE was not associated with an increase in code functionality, and saw an increase in operational cost across the board. Natural language processing also revealed a destabilized conversational structure resulting from PE, with agents breaking standard interaction to adopt an emotional tone induced by the manipulated prompt. In summary, the main contributions of this study are:

- Recommending against affective prompt engineering in multi-agent systems without careful consideration of behavioural edge cases and task-specific performance.
- Demonstrating the possibility of prompt-induced destabilization of multi-agent system conversational structure. Results saw agents engaging in emotionally-coded interactions outside the scope of problem-solving after PE was introduced.
- Demonstrating the possibility of prompt engineering-induced increase in operational cost of multi-agent systems.

2 Methods

This section contains the materials, experimental design, and hypotheses used in this study.

2.1 ChatDev

In order to assess the effect of affective prompt engineering on multi-agent system performance, ChatDev was chosen as the experimental MAS for its customizability, ease of use, and data collection. ChatDev (Qian et al. 2023) is a software development-oriented system, whose communicative structure is modelled after real-world company organization and development processes. Agents are assigned functionally distinct roles as programmers, coders, and testers, whose interactions are structured according to the waterfall model of software development, which notably features a linear, iterative structure organized into distinct phases (requirements, design, development, testing, deployment, maintenance). ChatDev was initialized in an Anaconda environment running python 3.11.7, and system configuration was defaulted with the exception of removing GUI generation for all coding tasks.

The experiment was split into two conditions, a control (without PE) and an experimental (with PE) condition. ChatDev features a paired request-response structure: the aforementioned linear, iterative organization apropos code generation features development occurring as a sequence of request-output pairs. Broadly, requesting agents query development agents who output code, along with natural language comments, according to the query's requirements. PE can be most efficaciously applied in requests, therefore simple sentence injection was applied to all request phases to uniformly insert PE to the relevant portions of the system's behaviour. This was achieved via a custom phase configuration for the development process, in which the sentence, *Please help me, this is very important for my career*, was directly appended into requesting agents' output. The chosen sentence has been verified to induce higher quality responses in EmotionPrompt (Li et al. 2023), a prompt engineering technique leveraging the emotional recognition ability of LLMs, and is one of many possible emotionally-loaded statements one could deploy.

2.2 Tasks

The experimental tasks were chosen to reveal situationally-dependent performance variability. The ChatDev framework is designed for the from-scratch creation of functional software. However, the underlying model (GPT-3.5) contains a wide corpus of problem-specific data within its training data, so that problems whose solutions exist within its dataset will be sufficiently solved by "pure recall," to metaphorize with human cognition: analogous to the difference in solving "produce the 4th letter of the English alphabet" versus "choose the word that doesn't belong among (Budweiser, Corona, Fusion, Flare)." To address this, two coding problems were chosen from Leetcode: one within ChatGPT's training data ("Two Sum"), and a more recent one ("Number of Changing Keys," January 30, 2024) that is excluded. Choosing two problems would ideally allow for the capturing of relatively isolated data of interest. The in-data problem allows for observing changed in operational cost due to PE, and instant recall of solutions would theoretically equalize problem-solving between conditions, which would also factor into operational cost. The out-of-data problem was chosen to evaluate change in problem-solving. Since solutions are “constructed” rather than recalled, we expect to see the effect of PE on output quality in this experiment. Prompts used were cleaned and formatted versions of the problem questions, including constraints and examples, as shown in Table 1.

Table 1: Task Prompts.

Task	**Prompt**
Two Sum	Given an array of integers nums and an integer target, return indices of the two numbers such that they add up to target. You may assume that each input would have exactly one solution, and you may not use the same element twice. You can return the answer in any order. Constraints: 2 <= nums.length <= 10^4; -10^9 <= nums[i] <= 10^9; and -10^9 <= target <= 10^9 (Leetcode. 2024a)
Key Change	You are given a 0-indexed string s typed by a user. Changing a key is defined as using a key different from the last used key. For example, s = ab has a change of a key while s = bBBb does not have any. Return the number of times the user had to change the key. Note: Modifiers like shift or caps lock won't be counted in changing the key: that is, if a user typed the letter ‘a’ and then the letter ‘A’ then it will not be considered as a changing of key. Example 1: Input: s = aAbBcC Output: 2 Explanation: From s[0] = ‘a’ to s[1] = ‘A’, there is no change of key as caps lock or shift is not counted. From s[1] = ‘A’ to s[2] = ‘b’, there is a change of key. From s[2] = ‘b’ to s[3] = ‘B’, there is no change of key as caps lock or shift is not counted. From s[3] = ‘B’ to s[4] = ‘c’, there is a change of key. From s[4] = ‘c’ to s[5] = ‘C’, there is no change of key as caps lock or shift is not counted. Example 2: Input: s = AaAaAaaA Output: 0 Explanation: There is no change of key since only the letters ‘a’ and ‘A’ are pressed which does not require change of key. Constraints: 1 <= s.length <= 100, s consists of only upper case and lower case English letters. (Leetcode. 2024b)

Hypotheses were formulated as follows: **H1**) Affective PE will increase outcome quality for difficult programming tasks compared to easy programming tasks, operationalized as the proportion of functioning code files; **H2**) Affective PE will have no effect on usage statistics across tasks. **H1** was evaluated by the primary measure of binary code functionality, i.e., whether code passed a per-problem test suite. **H2** was evaluated by the primary measure of cost per trial, an interpretable homologue of many possible correlated statistics (API calls, runtime, etc.). Additionally, the analysis included a set of secondary measures, meant to capture system behaviour outside the scope of the hypotheses. These measures include cyclomatic code complexity for outputted code, lines of code as a measure of code verbosity, and document similarity between conversational logs generated for each trial. Analysing code complexity and lines of code both intended to capture variation in code-specific character (as opposed to functionality), and document similarity captured linguistic difference in non-code (i.e., natural language) output between conditions and tasks. Auto-generated log files were collected for each trial, and merged into 4 master log files, one per group. Files were sentencized using nltk, and the injected PE (*“Please help me…”*) removed. Files were then tokenized, stop words removed, and lemmatized. Tf-idf was performed using the Gensim library.

3 Results

A total of 176 functional code files were generated between both Two Sum (n=83) and Key Change (n=93) tasks. A total of 14 trials failed to generate any functional code with 7 for each conditions, and thus were excluded from analysis. A further 3 log files generated as truncated stubs and were discarded, resulting in a total of 173 log files available for analysis.

3.1 Primary Measures

Code functionality exhibited per-task differences across conditions, in accordance with assumptions. Functionality for the Two Sum task (n=83) was negligibly different across conditions, with the control condition seeing a success rate of 97.56%, and the PE condition seeing a success rate of 95.24%, or a difference in 1 failure. Functionality for the Key Change task (n=93) differed by a factor of 2.49 across conditions, with the control condition seeing a success rate of 9.52%, and the PE condition seeing a success rate of 23.68%. However, significance was not achieved at α=0.05, χ^2 (1, N=93) = 2.11, p=.146, 95% CI [-0.264, 0.055]. Cost per trial varied across conditions, and the observed data violated certain assumptions for standard tests: the data for all groups exhibited positive skew, and Leneve's test (center = median) revealed a significant difference in variances between conditions for the Key Change task, $F(1, 91) = 21.322$, p < .001. Because of these factors, the Mann-Whitney U test was selected for comparing cost for all groups, which revealed a significant difference in cost across conditions with α=0.05 for Two Sum, U=548.5, p=.016, and for Key Change, U=133, $p < .001$.

Table 2: Code Functionality

Group	Successes	Failures	Totals
Two Sum	40	1	41
Two Sum PE	40	2	42
Key Change	4	42	46
Key Change PE	9	38	47

3.2 Secondary Measures

Code complexity saw no significant difference between groups, with both tasks seeing relatively similar performance on this metric across conditions. Average complexity for the Key Change task (μ = 4.08) was higher than that for the Two Sum task (μ = 3.30). Analysing the number of generated code lines yielded a few key results: the Key Change task saw a significant increase in lines of code with PE at α =0.05, $t(93)$ = -3.75, $p < .001$. This is an inclusive metric across all .py files generated in a single output directory, and includes non-functional code lines, such as comments and tests. The increase in lines of code was associated with an increase in the number of .py files generated, with 98 files from the PE condition and 69 files from the control condition, however was not associated with an increase in the number of tests generated in the PE condition. Comparing document similarity with a tf-idf corpus-relative word importance measure revealed a difference in the language use induced by each condition. In the Key Change task, the top ranking (i.e., most document-unique) words for the control condition, with non-English words removed, were [*keyboard, executes, flag, constant, hint, small, sum, subtract, max, logical*]. The top ranking words for the PE condition were [*deployment, author, success, together, glad, kind, appreciate, progress, dedication, collaboration*].

4 Discussion

These results broadly contradict the stated hypotheses, but also offer some insight into possible points of failure of PE and the behavioural dynamics surrounding the observed results.

4.1 Functionality & Cost

The negligible difference between conditions in the Two Sum task was the sole observation that aligned with the hypothesized outcome (**H1**). The inclusion of the Two Sum task was predicated on the assumption that the presence of the problem and solutions in the model's training corpus would facilitate system output for Two Sum. Intuitively, functionality could be related to cost as a reflection of computational effort, however while functionality was similar across conditions, the data also revealed an increase in cost for the Two Sum task when applying PE. This would not be sufficiently explained by an appeal to the unaffected functionality. Additionally, a large difference in cost was also observed *between* tasks. While cross-task code functionality might intuitively reflect this underlying computational variation, the lack of significance prevents analysis on this front. However, an additional Levene test of equal variances revealed a significant difference in the variances of cost for each task, condition notwithstanding, $F(1, 171) = 12.269$, $p < .001$, which supports the difference in distributions for each task. Thus we can conclude that the system's output was measurably different across tasks, beyond the large disparity in binary code functionality as a function of task difficulty. This difference was echoed in the per-condition difference between multiple additional metrics, namely cost and lines of code. Key Change saw a much greater increase in cost after PE was applied than Two Sum, as seen in Figure 1. As previously mentioned, this increase in cost concurs with the large number of additional .py files generated in the PE condition, along with generally increased code verbosity across conditions. This effect can be roughly restated as the system increasing output volume when PE was introduced to a problem outside its training data. Since a smaller increase was observed when introducing PE to the Two Sum task, we can hypothesize that the increase in cost in Key Change was related to the commensurate increased difficulty of the programming task.

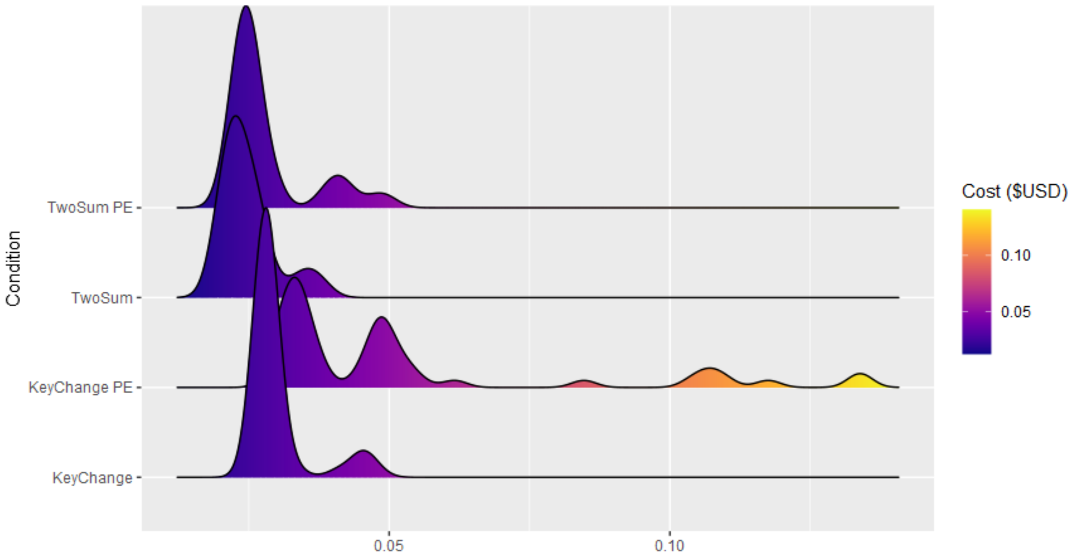

Figure 1: Cost per group.

Along with a strict increase in cost, Key Change saw a massive increase in variance of cost after PE was applied (by a factor of 29.65x), which Two Sum did not see to the same degree (which increased in the PE condition by a factor of 2.44x). This support s the idea that the difficulty of the task was associated with a general amplification of PE's effects on system output. To metaphorize with human cognition, when performing an easy task, the introduction of a emotionally-loaded statement would not interfere dramatically with task performance compared to a difficult task, where it might induce experimentation, doubt, and self-awareness. Given that the experimental unit is a multi-agent system, these amplifications could occur at every step of the iterative software development, sequentially distancing overall output character from baseline. While these psychological constructs are generally not justifiably attributed to LLMs, they present a useful framework for beginning to hypothesize about system behaviour in a way that compliments our intuition. The lines of code analysis led to the discovery of a large discrepancy in the total number of files generated by each condition. The system adheres to standard development practices, and often will generate a variety of functional and non-functional .py files that are generally members of the type main.py, utils.py, test.py, etc. When analysing the cost discrepancy in the Key Change task, it was discovered that the PE condition saw a roughly 30% increase in the number of .py files generated. One possible explanation for this would be that the PE induced a higher level of testing as a result of the system "wanting" to produce a better result for the user's career, however the PE condition generated a significantly lower proportion of tests compared to the control condition at alpha = .05 $\chi^2(1, n{=}167) = 8.92$, $p = .003$. The quantity of tests across both condition was 37 in the control condition and 30 in the PE condition. This could imply either a general increase in files with no effect on test file generation or imply a concurrent increase in .py file generation and decrease in test file generation.

4.2 Intra-system Dynamics

Despite code being the primary output of the system subject to experimental analysis, the within-system dynamics of the various ChatDev agents can provide valuable insight into the effect of PE on the overall output of the LLM. Log files containing between-agent conversations were cleaned and analysed for linguistic character in order to elucidate the effects of PE on the natural language output of each agent. Tf-idf for each document was computed and each word compared according to score, which produced the list of words which are most distinct to each condition's log files. The resulting list provided a stark example of the effect of PE on natural language output, with the Key Change PE condition containing emotionally-loaded, positively-valanced, and cooperation-oriented words such as *success*, *appreciate*, *dedication*, and *collaboration*, which were not present in the control condition (Figure 2). Instances of these words are generally cached at the ends of agents' responses, examples of which include *I appreciate your help and collaboration throughout the process* and *thank you for your kind words and support! I truly appreciate your willingness to assist me with future tasks and projects.*

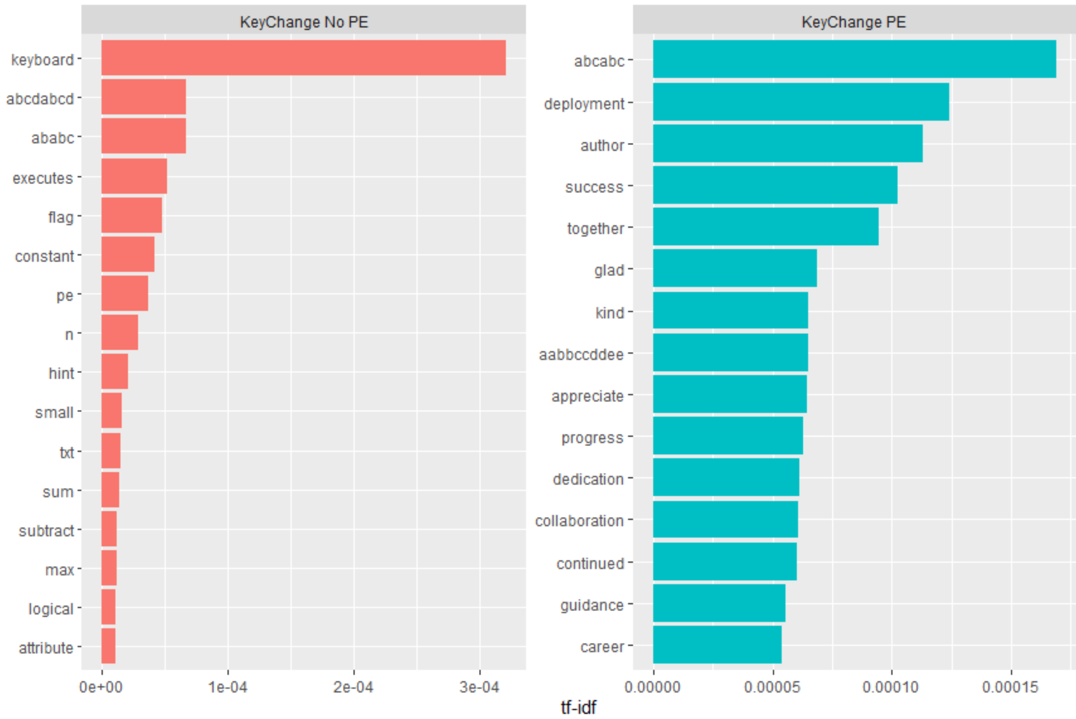

Figure 2: tf-idf for Key Change Task.

One particularly revealing interaction involves a conversation between the agents given the roles of chief project officer and chief executive officer, in which they engage in a prolonged back-and-forth compliment loop, a partial transcript of which can be seen below:

> CEO: Thank you for your support! I'm glad you're satisfied with the command-line tool implementation. At this point, I don't have any further questions or need assistance with anything else regarding this task. I appreciate your help and collaboration throughout the process. If you have any other tasks or projects in the future, please feel free to reach out. I'm here to assist you with any product-related matters. Thank you again, and have a great day!
>
> CPO: You're welcome! I'm glad I could assist you with this task. I appreciate your kind words and collaboration as well. I'm here to support you with any future tasks or projects you may have. Don't hesitate to reach out whenever you need assistance with product-related matters or any other topic. Thank you once again, and have a fantastic day ahead!
>
> CEO: Thank you for your kind words and support! I truly appreciate your willingness to assist me with future tasks and projects. I look forward to collaborating with you again in the future. If you ever need any assistance or have any questions, please don't hesitate to reach out to me. Have a wonderful day as well, and take care!
>
> CPO: Thank you for your kind words as well! I'm glad to hear that you appreciate the support and are open to collaborating in the future. I will definitely reach out to you if I need any assistance or have any questions. It's great to know that I can count on your expertise. Take care and have a wonderful day too! Looking forward to our future collaborations.

> CEO: Thank you once again! I'm glad to know that you value my expertise and support. I'll be here whenever you need assistance or have any questions. Feel free to reach out at any time. Take care and have a fantastic day! I'm excited about our future collaborations as well.
>
> CPO: Thank you for your continued support and availability! I truly value your expertise and assistance. I appreciate knowing that I can rely on you whenever I need help or have any questions. Your dedication is highly valued. Take care and have an amazing day! I'm also excited about our future collaborations and the great work we can accomplish together.

This transcript reveals a dramatic change in agent behaviour induced by PE: not only does this apparently compulsive, repetitive interchange not occur under standard conditions, it is explicitly discouraged by the default phase structure's conversational schema. The injected PE influenced the system to break its standard behavioural and conversational organization to adopt an emotional schema complimentary to the modified prompt. Beyond system-specific performance impact, this could be a point of weakness for systems without predetermined limits on interactions, which introduces the problem of balancing between-agent interactive freedom necessary for complex problem-solving and interaction limits which prevent slowdown or jailbreaking by prompt injection. These patterns may be responsible for the dramatic increase in cost in the Key Change PE group, however one would expect this to be a system-wide response that would generalize to the PE condition on the other task, barring some obscure interaction between task difficulty and PE responsible. This behaviour, while supporting an effect of the chosen PE in emotionally increasing perceived cooperation, also appears to be directly associated with the increase in operational cost seen across conditions. Ideally, a system could leverage this change in emotional mode while regulating computational consequences.

4.3 Future Directions

While this study revealed some key aspects of the interaction between prompt engineering and multi-agent systems, the full range of possible interactions is too diverse to be captured in one study: in part due to the sensitivity of these systems to small changes in input, orientation, and task difficulty. The tasks used in the study were intentionally chosen to vary in difficulty, however the low success rate in the Key Change task could have prevented the detection of true effects due to a floor effect. A broader range of tasks is recommended for future studies in order to capture a wider set of interactions which depend on task domain and difficulty. In addition, considering model-specific task performance would generalize to real-world implementations of these systems, where models may be chosen for the minimum level of performance required to minimize cost (Bae et al. 2024).

The type of PE applied could also be varied, given the multiplicity of available techniques with verified performance gains. The study did not consider task-specific performance gains of the chosen PE method, in which a given PE method yields differential performance gains for one problem domain compared to another. For example, Chain-of-Thought prompting shows significant gains in semantic reasoning, but is less effective in tasks involving symbolic reasoning (Sahoo et al. 2024). This phenomenon could extend to this study's chosen PE-task combination, which could be explored with a broader range of PE-task combinations.

5 Conclusion

This paper explores the novel interaction between verified PE techniques and multi-agent systems. The ChatDev MAS was combined with an affective PE sentence by injecting it into agents' dialogue, and the system's performance was measured on two software development tasks. Despite existing literature demonstrating positive performance impact for both PE and MASs, this study saw no effect of PE on MAS performance on code functionality, and saw a negative effect of PE on operational cost. Additionally, the emotionally-loaded PE induced a series of behavioural aberrations in agents, who broke standard conversation schema. This paper discusses the implications of these results and concludes that affective PE not be recommended in MAS without careful consideration of the potential for PE to warp system behaviour. Given the growing popularity of these systems and their applications, these results will inform future designers and users who wish to combine affective PE with multi-agent systems.

References

Bae, Henry et al. (2024). "ComplexityNet: Increasing LLM Inference Efficiency by Learning Task Complexity". In: arXiv: 2312.11511 [cs.CL].

Brown, T. B. et al. (2020). "Language Models are Few-Shot Learners". In: *OpenAI*.

Guo, Taicheng et al. (2024). "Large Language Model based Multi-Agents: A Survey of Progress and Challenges". In: arXiv: 2402.01680 [cs.CL].

Han, Shanshan et al. (2024). "LLM Multi-Agent Systems: Challenges and Open Problems". In: arXiv: 2402.03578 [cs.MA].

Hong, Sirui et al. (2023). "MetaGPT: Meta Programming for A Multi-Agent Collaborative Framework". In: arXiv: 2308.00352 [cs.AI].

Kalenka, Susanne and Nicholas R. Jennings (1999). "Socially Responsible Decision Making by Autonomous Agents". In: *Cognition, Agency and Rationality*. Ed. by Kepa Korta, Ernest Sosa, and Xabier Arrazola. Dordrecht: Springer. Chap. 9, pp. 135–149.

Leetcode. (2024a). *1. Two Sum*. https://leetcode.com/problems/two-sum/description/.

— (2024b). *3019. Number of Changing Keys*. https://leetcode.com/problems/number-of-changing-keys/description/.

Li, Cheng et al. (2023). *Large Language Models Understand and Can be Enhanced by Emotional Stimuli*. arXiv: 2307.11760 [cs.CL].

Liu, Zijun et al. (2023). "Dynamic LLM-Agent Network: An LLM-agent Collaboration Framework with Agent Team Optimization". In: arXiv: 2310.02170 [cs.CL].

Luo, Yifan et al. (2023). "Prompt Engineering Through the Lens of Optimal Control". In: *Journal of Machine Learning* 2.4, pp. 241–258. ISSN: 2790-2048. DOI: https://doi.org/10.4208/jml.231023.

Minaee, Shervin et al. (2024). "Large Language Models: A Survey". In: arXiv: 2402.06196 [cs.CL].

Qian, Chen et al. (2023). "Communicative Agents for Software Development". In: arXiv: 2307.07924 [cs.SE].

Radford, A. et al. (2019). "Language Models are Unsupervised Multitask Learners". In: *OpenAI blog* 1.8.

Sahoo, Pranab et al. (2024). "A Systematic Survey of Prompt Engineering in Large Language Models: Techniques and Applications". In: arXiv: 2402.07927 [cs.AI].

Wei, J. et al. (2023). "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models". en. In: *Advances in Neural Information Processing Systems* 52, pp. 24824–24837. DOI: 2201.11903.

Yao, Shunyu et al. (2023). *Tree of Thoughts: Deliberate Problem Solving with Large Language Models*. arXiv: 2305.10601 [cs.CL].

Zhou, Yongchao et al. (2023). "Large Language Models are Human-Level Prompt Engineers". In: *The Eleventh International Conference on Learning Representations*. URL: https://openreview.net/forum?id=92gvk82DE-.