Towards Improving Visual Chatbot Builders with Intent-based Copilots

Emanuel Lacic[1], Petra Fribert[1], Anamarija Lukac[1] and Ivica Lovric[1]

[1]*Infobip, Zagreb, Croatia*

Abstract

Modern visual-based chatbot building solutions have the aim to simplify the development and deployment of sophisticated and responsive conversational agents. But such an environment sometimes presents a steep learning curve when balancing the complexity of available building features with an intuitive UI design. The recent rise in popularity and availability of LLMs has opened up new possibilities in helping users in their experience by introducing intent-based copilots in the underlying platform. In this study, we explore the impact of fine-tuning existing commercial and open-source models to generate the building blocks of a conversational agent. In order to asses the performance of such copilots, we investigate their accuracy, bias towards popularity and the amount of hallucinations that a particular generative strategy may exhibit. We report the performance differnces between purely prompt-based strategies and fine-tuned models. Moreover, we also show that smaller models which require less hardware resources can exhibit a similar performance when compared to their larger counterparts.

Keywords

Visual Chatbot Building, Large Language Models, Intent-based Copilots, Accuracy, Popularity Bias, Hallucinations

1. Introduction

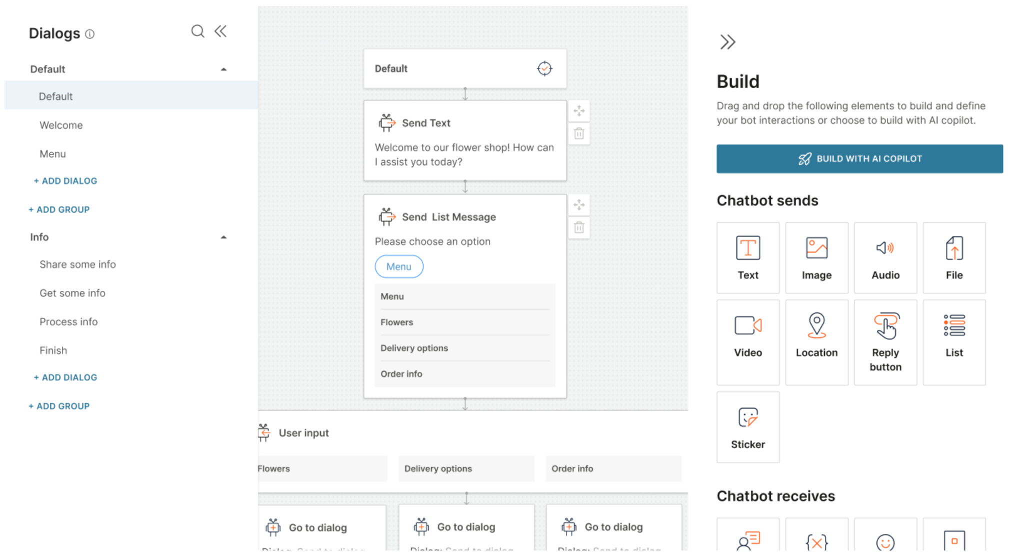
Modern visual-based chatbot building solutions, similar to platforms like Dialogflow [1] or Flow XO [1], aim to make the way chatbots are developed and deployed as simple as possible. They typically offer a user-friendly interface for designing conversation flows, focus on intents and contexts rather than linear scripting and allow for more natural and dynamic user interactions. A critical aspect of these platforms is their ability to manage multiple dialogs. A single chatbot is often composed of numerous dialogs, each tailored to specific topics or functions. This modular approach allows chatbots to handle a wide range of user queries and maintain coherent conversations over various subjects, thus enhancing customer engagement and automating interactions efficiently. A key priority here is to provide an optimal user experience and as such, it comes only natural that there is a growing trend to provide an intent-based copilot functionality [2]. The term "copilot" in this sense describes a functionality that translates a textual user intent into an intent-based prediction which is then transformed into a suitable format for user interaction [3]. For example, Github's Copilot X [2] uses the power of Large Language Models (LLMs) to offer a conversational interface that allows programmers to express their intent in order to generate or modify code.

Since the introduction of Transformers [4], there has been a rise in the availability of LLMs, both commercial (e.g., OpenAI [5] or Google [6]) and open-source ones (e.g., Mistral [7]). With such an increased availability of different models, it becomes much easier to integrate the output of LLMs into software applications and create copilot functionalities. This has already sparked interest in exploring the challenges when integrating LLMs with conversational interfaces. For example, [8] have investigated trust, user experience and different evaluation metrics when building an LLM-based assistant deployed at Meta. In [9], the authors present a systematic design exploration of user interfaces for AI-driven code editors. Others [10, 11] look into different tasks like video editing and investigate interfaces for

Joint Proceedings of the ACM IUI Workshops 2025, March 24-27, 2025, Cagliari, Italy.

emanuel.lacic@infobip.com (E. Lacic); petra.fribert@infobip.com (P. Fribert); anamarija.lukac@infobip.com (A. Lukac); ivica.lovric@infobip.com (I. Lovric)
0000-0002-3059-0502 (E. Lacic)

[1]https://flowxo.com/
[2]https://github.com/features/preview/copilot-x



Welcome to our flower shop! How can I assist you today?

Please choose an option

Drag and drop the following elements to build and define your bot interactions or choose to build with AI copilot.

Figure 1: Visual chabot building interface at Infobip. One chatbot consists of multiple dialogs with relationships between themselves (left). A user can define individual visual building elements (right) within a particular dialog (center) to define the intended behavior.

multiple modalities of input. With respect to building copilots, [3] state that a major pain point lies in prompt engineering and testing, as these are extremely time-consuming and resource-constrained tasks. Another issue may also be the need for an adaptive interaction experience as shown by [12]. The authors reported a user study where participants would often express frustration with the copilot due to usability issues, particularly when unsolicited suggestions slow down their workflow. As such, in this work we contribute to the scarce research on utilizing LLMs when creating a copilot functionality and apply it to the problem of visual chatbot building at Infobip [3]. Here we explore the usage of four different prompting techniques when used with commercial models from OpenAI and Google, as well as fine-tune four LLMs which vary in different sizes with respect to model parameters. Our results not only shed light into the trade-off between the accuracy performance and amount of hallucinations, but also highlight the bias towards popularity which differs between the respective generation strategies. Finally, we look into the potential effect of position bias when using a generative copilot in the UI and discuss a model selection approach in contrast to letting the user choose the underlying model.

2. Problem Definition

As seen in Figure 1, a visual chatbot builder at Infobip consists of one or more elements within a dialog. Each element here represents a building block like a text message to the user, an API call or a condition that branches the flow, among others. The aim of this paper is thus to investigate how good can we predict the elements of a given dialog in the presence of a textual input that states the user's desired intent of what that particular dialog should be doing. As such, we define $S = \{E_1, E_2, \ldots, E_N\}$ as the vocabulary, i.e., the set of all possible visual chatbot-building elements. An element x in the list that we wish the copilot to generate can either be a particular element from S or a list of visual elements from S. Formally, $x \in S$ or $x \subseteq S$. The latter is needed to encode the ordering of branching elements like a conditional block that will lead to a different behavior depending on what kind of condition is met. The entire generated list structure L can then be defined as a list of such elements and is represented as $L = [x_1, x_2, \ldots, x_n]$, where each x_i is either a member of S or a subset of S, and n is the length of the list L. By setting the copilot prediction problem in such a way, we aim to investigate what LLM-based generation strategy G_s is best suited to generate a valid list L that can be rendered in the UI and does actually achieve the desired intent of the dialog. That is, we explore solving $G_s(p) = L$, where p is a textual prompt that describes the desired intent of a chatbot's dialog.

[3]https://www.infobip.com/answers

3. Dataset

To evaluate the efficacy of our approach, we utilize a proprietary dataset that consists of $2,084$ English speaking and actively used chatbots. The dataset was anonymized and cleaned from any sensitive information which resulted in $59,429$ individual dialogs. Overall, there are 25 different visual building elements that serve their purpose in building the sequential flow of how the chatbot should behave (e.g., sending a text message, calling an API, etc.). The maximum number of visual chatbot building elements a dialog in our dataset has is 15 and on average there are 5.81 elements in a dialog. Interestingly enough, we found that while building chatbots with visual elements there are some building blocks which occur quite frequently. For example, the element which is directing the flow of the chatbot from one dialog to another accounts for 38% of all element occurrences within the dialogs which suggests that there is an inherent bias towards popularity in the data. For each dialog we have a textual description or rather the input prompt about its intent as described in the previous section. This textual prompt, for example, is in the form of "A dialog that helps users check the working hours and directs them to a call center or waits for an agent to contact them", "End a customer's session and ask for a feedback survey" or "Check current account balance via WhatsApp". On average, an input prompt contains about 22 words, where the longest dialog intent description had 57 words and the shortest consisted of 6 words.

4. Experimental Setup

As already mentioned, the aim of our work is to investigate how good can generative models create a dialog flow when given an input textual prompt which describes the intent of the dialog. To do so, we first experiment with four different prompting strategies using commercially available APIs from OpenAI [5] and Google [6] as using pre-trained models in such a way has become a de-facto standard for many software solutions that aim to integrate the generative capability of LLMs. Secondly, we also look into fine-tuning four LLMs that vary in different sizes using our own data. Due to the limited availability of hardware resources we had for fine-tuning existing LLMs, we split the dataset randomly into a train, validation and test set instead of using a k-fold cross-validation. That is, we sampled 20% or rather $11,886$ dialogs to be used as the test set. For the remainder we also employed a $80/20$ split which resulted in a train set of $38,034$ dialogs and a validation set of $9,509$ dialogs. For all experiments, we investigate the temperature setting for a more deterministic output of the generative model (i.e., a temperature of 0.0) as well as a more exploratory one (i.e., a temperature of 0.7).

4.1. Prompt Baselines

A combination of tuning-free prompting and prompt augmentation (i.e., in-context learning) is nowadays a common approach for generative models as it can produce efficient results without the need of updating parameters in the underlying LLM [13]. As such, we investigate different in-context learning prompting strategies on OpenAI's GPT-3.5 Turbo [5] and Google's Gemini 1.0 Pro [6] models [4]. We first explore zero-shot prompting with only a natural language description of the task at hand. This is then extended by a few-shot prompting strategy, where additional demonstrations of the task are given as conditioning at inference time [14]. Here we also investigate two levels of granularity when providing task demonstrations with just examples of the input and output as well as additional rules. Finally. we investigate Chain-of-Thought [15], a prompting strategy that improves the ability to perform complex reasoning through intermediate reasoning steps. Summed up, in our experiments we make use of the following four prompt strategies:

Prompt strategy #1: Zero-Shot. For a zero-shot setting we define a prompt that describes the problem of building a chatbot dialog as well as states the vocabulary of the available 25 visual elements. The model was further instructed to produce the expected array L of elements from the vocabulary based on the added textual dialog intent (i.e., task from the test set).

[4]These were the two commercially available LLMs that were most popular at the time of conducting the experiments.

Prompt strategy #2: Few-Shot Example. We extend the zero-shot prompt by adding three different examples of input task descriptions and their expected output.

Prompt strategy #3: Few-Shot Rule. In addition to providing few examples of a valid input and output in the prompt, we add the information about specific rules that need to be enforced in order to render the generated prediction in the UI. These rules are highly domain-specific and contain, for example, a restriction where some elements can't have multiple instances within a single dialog. Certain elements are also mutually exclusive and cannot coexist in the same dialog. Additionally, some elements function as branching points, and their subsequent branches are limited in size. Overall, we include 7 different rule examples which need to be ensured for a proper rendering of dialog elements.

Prompt strategy #4: Chain-of-Thought (CoT). Finally, we utilize a more robust technique that is designed to enhance the reasoning ability of an LLM by instructing it to generate a series of intermediate steps that lead to the final answer. These intermediate steps, known as the chain of thought, have been shown to significantly improve the model's ability to perform complex reasoning [16, 17]. The effectiveness of CoT is particularly effective with larger and more powerful language models, as it is an emergent property of model scale [15]. As CoT prompting has proven effective in improving the performance of LLMs across various reasoning tasks, we extend the previous prompting strategy with 5 reasoning steps to (1) understand the input, (2) identify the main actions, (3) select and sequence the appropriate visual elements, (4) create branching logic if needed, and (5) ensure that the dialog sequence ends properly. For the three input and output examples, a rationale (i.e., explanation of the reasoning steps) has been added.

4.2. Fine-Tuned Models

To find out how good we can model the domain specific knowledge of building chatbots using visual elements, we fine-tune existing LLM solutions using our own data. Here we investigate four different LLMs which range from a large parameter size to a much smaller size that is better suited for commodity hardware. The largest model is the commercially available GPT-3.5 Turbo for which we use Microsoft's API [5] to fine-tune a custom model. For the open-source ones, we apply Low Rank Approximation (LoRA) [18] which introduces a pair of rank decomposition matrices that are trained simultaneously while keeping the existing weight matrices fixed. As the number of trainable parameters is determined by the rank r, we set it to 4 as such a number was shown to be sufficient [18]. Overall, we experiment with the following four LLMs that we fine-tuned using our training dataset:

GPT-3.5 Turbo (large) [5] is the largest model in our experiments. The exact model size is not publicly known but its predecessor GPT-3 [14] is reported to have 175B parameters (i.e., about 10 times more than GPT-2). Although there are some estimates in the community that GPT-3.5 Turbo is much smaller in size when compared to GPT-3, it is by far still the largest model in our experiments. We opted for fine-tuning this model due to its reported size as well as popularity in both the academia and industry.

Mistral-7B (mid) [7] is a recently popularized 7B parameter language model as it exhibited a better performance than LLaMa2-13B across multiple benchmarks as well as outperformed the original LLaMa-34B model in areas like reasoning, mathematics, and code generation. It employs grouped-query attention to accelerate inference speed and reduce memory requirements during decoding. Additionally, sliding window attention is employed to handle longer sequences more effectively at a reduced computational cost.

LLaMa-3B [19] (small) utilizes a modified transformer architecture for enhanced performance and is designed for efficient memory and runtime usage. The training data, drawn from sources like CommonCrawl, GitHub, and Wikipedia, is processed with byte-pair encoding to ensure quality and variety. LLaMA-3B has become a popular choice for a smaller model as it demonstrate competitive performance across various tasks, including zero-shot and few-shot challenges. We opted out for the

[5]https://learn.microsoft.com/en-us/azure/ai-services/openai/tutorials/fine-tune

original version of the smaller LLaMa model as it requires less hardware resources (e.g., when only commodity GPUs with a small VRAM size are available). Furthermore, we were interested in the performance when we have a 2k context length restriction.

Sheared-LLaMa-1.3B [20] (tiny) is constructed by successfully pruning LLaMA2-7B [21] down to 1.3B parameters, while outperforming equivalent-sized models in various downstream tasks. This is done by using two key techniques: targeted structured pruning, which reshapes a larger model by pruning layers, heads, and dimensions, as well as dynamic batch loading, which adjusts training batch composition based on varying losses across different domains. Having such a tiny LLM be able to perform reasonably well for the task of generating dialog elements would lead to the least expensive option for running and scaling the copilot functionality when needed.

Inference Considerations. We fine-tuned the open-source LLMs on a GPU server with a NVIDIA Tesla V100 (16 GB VRAM). For inference, we used Huggingface's text generation API [6], compatible with LLaMa-3B and Sheared-LLaMa-1.3B. However, running Mistral-7B on NVIDIA's Volta architecture required the use of llama.cpp [7]. Mistral-7B needed 13.6 GB of VRAM, compared to 9.5 GB for LLaMa-3B and 5.1 GB for Sheared-LLaMa-1.3B.

4.3. Metrics

To asses the performance of our four prompting strategies on GPT-3.5 Turbo and Gemini 1.0 Pro as well as the four fine-tuned LLMs, we measure both the prediction accuracy as well introduce a score to measure the amount of hallucinations that a respective model generates.

nDCG is a ranking-dependent metric that measures how many visual elements within the chatbot builder are generated correctly. It also takes the position of the elements in the predictions into account where it favors more when the correct elements are generated at the beginning. It is calculated by dividing the discontinued cumulative gain (DCG) of the predicted list of elements with the ideal DCG value, which is the highest possible DCG value that can be achieved if all the relevant elements would be generated in the correct order [22]. In our case, having a high nDCG is favourable for ensuring a better user experience. For example, if the model has generated a valid sequence of visual elements at the beginning but hallucinated at the end, we may want to render at least the first part of the generated sequence in the UI instead of showing an empty dialog.

Recall is calculated as the number of correctly generated visual elements divided by the number of relevant elements in the respective test case. Optimizing for a higher recall would mean that we aim to better encapsulate the desired intent which was stated in the input prompt.

HitRate is measured as 1 when the list L of generated visual elements matches by 100% the list of elements which is expected, else it is 0. That is, we are interested to uncover how good can the generative models fully reproduce the expected behaviour of the chatbot dialog.

Hallucination Score is based on the presence or absence of wrongly generated responses (i.e., hallucinations). It is assigned a value of 1 if there are no hallucinations, otherwise, it is 0. A hallucination in this context is identified when any of the following three validations are violated: (1) format validation, (2) vocabulary validation and, (3) rule validation. The format validation requires that the generated text is a parsable list as defined in Section 2 without any additional free-form text in it. The vocabulary validation ensures that only the available chatbot elements defined in the vocabulary of the training set are generated and no other generalization is happening. Lastly, the rule validation checks that the visual elements adhere to the predefined set of domain-specific rules regarding placement and user interface rendering as described in the third prompting strategy of the previous Section 4.1 (i.e., the Few-Shot Rule prompting strategy).

[6]https://github.com/huggingface/text-generation-inference
[7]https://github.com/ggerganov/llama.cpp

		nDCG ↑	Recall ↑	HitRate ↑	Hallucination Score ↓
Temperature = 0.0					
GPT-3.5 Turbo	Zero-Shot	0.5154	0.3085	1.31%	30.67%
	Few-Shot Example	0.6402	0.3448	2.13%	20.22%
	Few-Shot Rule	0.6201	**0.3901**	0.68%	23.57%
	CoT	0.6412	0.3242	3.60%	8.68%
Gemini 1.0 Pro	Zero-Shot	0.6073	0.3686	**5.89**%	91.85%
	Few-Shot Example	0.7570	0.3469	4.88%	54.47%
	Few-Shot Rule	**0.7993**	0.3682	1.93%	83.85%
	Chain-of-Thought	0.7416	0.3675	5.35%	**8.38**%
Temperature = 0.7					
GPT-3.5 Turbo	Zero-Shot	0.5085	0.3131	2.09%	46.44%
	Few-Shot Example	0.6228	0.3398	1.75%	12.63%
	Few-Shot Rule	0.5940	**0.3885**	0.69%	25.70%
	CoT	0.6459	0.3196	3.68%	9.17%
Gemini 1.0 Pro	Zero-Shot	0.6271	0.3547	5.00%	92.60%
	Few-Shot Example	0.7607	0.3316	3.84%	64.73%
	Few-Shot Rule	**0.7725**	0.3563	1.84%	91.77%
	CoT	0.7292	0.3638	**5.41**%	**9.06**%

Table 1
Accuracy and hallucination performance of the four prompt baseline strategies with temperature settings of 0.0 and 0.7 using the commercially available API for OpenAI's GPT-3.5 Turbo and Google's Gemini 1.0 Pro. Bold numbers indicate the highest performing model of a particular metric and temperature.

5. Accuracy Performance

The accuracy performance with respect to nDCG, Recall and HitRate for the prompt baseline strategies can be seen in Table 1 and for the fine-tuned LLMs in Table 2.

Prompt Baselines. When comparing GPT-3.5 Turbo and Gemini 1.0 Pro, an interesting observation can be seen across both temperature settings. Namely, with respect to nDCG and HitRate, Gemini 1.0 Pro produces more accurate predictions when comparing the individual prompting strategies between themselves. In the case of Few-Shot prompting, however, GPT-3.5 Turbo did manage to achieve the best Recall. When looking at the individual prompting strategies, a Few-Shot prompting strategy can result in a better accuracy than employing a Zero-Shot one. However, adding rules to be enforced in the Few-Shot prompting strategy did negatively impact the HitRate. But interestingly enough, this effect can be countered by applying Chain-of-Thought. Using a CoT strategy on GPT-3.5 Turbo has consistently resulted in the best performance with respect to HitRate and nDCG. When applied to Gemini 1.0 Pro it also showed comparable results (i.e., even having the overall best HitRate of 5.41% for the temperature setting of 0.7), clearly indicating it to be a robust prompting technique when applied for the task of predicting the sequence of elements for visual chatbot-building solutions.

Fine-Tuned Models. One of the goals of our investigation was to find out how much can we improve the accuracy performance when we fine-tune LLMs by applying the LoRA on our data. As seen in Table 2, the fine-tuned LLMs outperform the prompt baseline strategies with respect to all three accuracy metrics. Mistral-7B had the most notable HitRate of 26.96%, while the fine-tuned version of GPT-3.5 Turbo had the best overall Recall of 0.4327. Interestingly enough, Sheared-LLaMa-1.3B which was the smallest fine-tuned model in our experiments achieved the best nDCG score of 0.8920. Moreover, we found that decreasing temperature, the randomness controlling hyperparameter, from 0.7 to 0.0 mostly increased the accuracy performance in all models and baselines except in GPT-3.5 Turbo and Mistral-7B.

6. Reduction of Hallucinations

The biggest downside when utilizing prompting strategies is shown in Table 1 with the Hallucination Score. Ideally, we are interested to see a number that is close to 0 as possible. A high hallucination score indicates a poor user experience when applying the copilot in the UI as it often prevents the generated dialogs from being rendered, which in turn can cause frustrations to the end-user as reported by [12].

	nDCG ↑	Recall ↑	HitRate ↑	Hallucination Score ↓
Temperature = 0.0				
GPT-3.5 Turbo (large)	0.8903	**0.4327**	15.78%	1.96%
Mistral-7B (mid)	0.8848	0.4275	**26.72%**	15.34%
LLaMa-3B (small)	0.8859	0.4248	18.89%	0.19%
Sheared-LLaMa-1.3B (tiny)	**0.8920**	0.3948	18.01%	**0.04%**
Temperature = 0.7				
GPT-3.5 Turbo (large)	**0.8903**	**0.4327**	15.78%	1.88%
Mistral-7B (mid)	0.8854	0.4276	**26.96%**	15.16%
LLaMa-3B (small)	0.8515	0.4112	16.77%	0.66%
Sheared-LLaMa-1.3B (tiny)	0.8543	0.3776	14.56%	**0.12%**

Table 2
Accuracy and hallucination performance of the fine-tuned large, mid, small and tiny LLM with temperature settings of 0.0 and 0.7. The bold numbers indicate the highest performing model of a particular metric and temperature setting.

Prompt Baselines. A notable difference can be seen between GPT-3.5 Turbo and Gemini 1.0 Pro when we compare them using the Zero-Shot and Few-Shot prompting strategies. Altough, Gemini 1.0 Pro exhibited a much better accuracy performance as reported in the previous Section 5, it produced a significantly higher number of hallucinations. This is especially the case for a Zero-Shot strategies where it hallucinated over 90% of the time. Although the performance with respect to hallucinations seems to fluctuate quite a lot between the individual prompting strategies, applying Chain-of-Thought in our case does lead to a more robust and consistent performance between the two utilized commercially available LLMs. To be concrete, applying a CoT strategy resulted in the lowest hallucination score out of all prompt baselines with 8.38%.

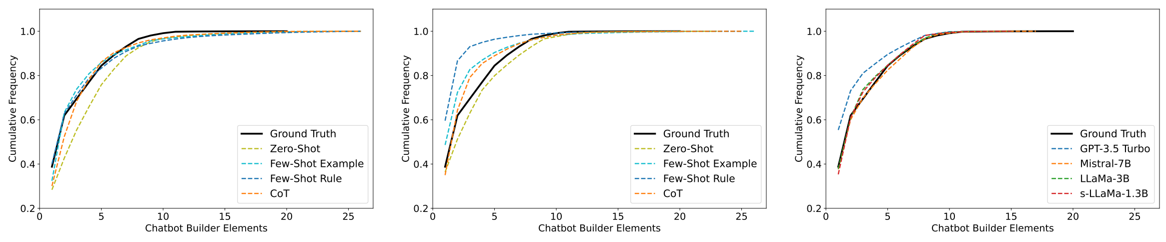
Fine-Tuned Models. By looking at the amount of hallucinations, we can see the biggest value of fine-tuning LLMs in the context of building an intent-based copilot for visual chatbot builders. Most of the hallucinations in all LLMs included the generation of visual element types that do not exist in our application domain or are a result of invalid textual formatting of valid visual element types. The latter, for instance, includes lowercasing characters that should have been in uppercase or not following the expected structure as defined in Section 2. To our surprise, Mistral-7B, which was the best performing model with respect to HitRate, resulted in a lot more hallucinations when compared to other fine-tuned LLMs, which is not favorable when needing to provide a copilot functionality in a real setting. Actually, it hallucinated significantly more than the CoT prompt strategies that were run on both GPT-3.5 Turbo and Gemini 1.0 Pro. Upon analyzing the responses of Mistral-7B, we have discovered that the biggest problem was in generating a response in the expected textual format, often resulting with empty outputs containing no visual elements that could be rendered.

The most notable finding however was the performance of the smaller Sheared-LLaMa-1.3B model. It was not only the smallest fine-tuned LLM that we employed, but also resulted in the best overall performance with respect to hallucinations. In both temperature settings it achieved the lowest amount hallucinations and managed to reach a score of 0.04% hallucinations overall. This is especially interesting when we consider that it achieved a comparable accuracy performance to other fine-tuned models while being a cost effective solution that can be efficiently run on commodity hardware.

7. Bias Analysis

In order to further uncover the effects of intent-based copilots in the context of visual-based chatbot building, we also explore the amount of bias towards popularity as well as explore the effect of position bias when giving users the choice of specifying which model should be used.

Impact of Popularity Bias. Having a bias towards popularity [23] is known to lead to a lack of diversity in generated content and, in our case, potentially cause an unfair exposure for less popular visual elements. By assessing and mitigating this kind of bias, we are fostering a more diverse and



(a) Prompt Baselines - GPT-3.5 Turbo (b) Prompt Baselines - Gemini 1.0 Pro (c) Fine-tuned LLMs

Figure 2: Popularity bias of prompting baselines on GPT-3.5 Turbo (left), Gemini 1.0 Pro (center) and fine-tuned models (right) when compared to the ground truth data from the test set (black line).

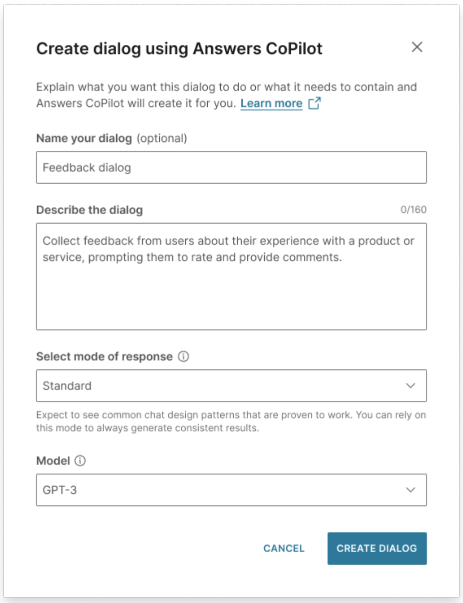
personalized copilot functionality. As such, in Figure 2 we quantify the extent to which the LLM-based element generations deviate from the established ground truth of the popularity like in [24].

In our experiments, only 20 from the available 25 visual elements were part of the test set. It can also be seen that there is one element which is by far the most popular one and accounts for 38% of all element occurrences within the dialogs. By utilizing only prompting techniques without any fine-tuning we end up with a generated list of elements that is less biased towards popularity. In addition to that, we can see that other visual elements, which were not available in the test set, also got the chance to be generated and rendered in the UI. Out of all evaluated approaches, the Zero-Shot prompt strategy had the lowest bias towards popularity for both GPT-3.5 Turbo and Gemini 1.0 Pro. Interestingly enough, by using Gemini 1.0 Pro we end up with generated dialogs which favor much more the popular visual elements from the test set. Out of the three strategies that included examples of a valid input and output in the prompt, the CoT technique not only achieved a better performance with respect to accuracy and hallucinations but also manages to be less prone to popularity bias when compared to the other two Few-Shot strategies. By fine-tuning LLMs with our training data we end up with a similar bias towards popularity when compared to the ground truth dialog elements from the test set. Moreover, the fine-tuned models cover only 17 of the available 25 dialog elements. Interestingly enough, the fine-tuned version of GPT-3.5 Turbo can be seen to exhibit a much higher bias to popularity as the most popular element accounts for 55% of the occurrences in all generated lists of elements.

Position Bias on Model Selection. To assess the usability of the fine-tuned models in a copilot setting, we also conducted a small qualitative analysis with 20 internal participants (i.e., expert users that were familiar with the platform) who generated 200 new chatbot dialogs. The participants could access the chatbot building platform and create a new dialog as seen in Figure 3 by submitting the desired textual input. Additionally, the participants were instructed that they had the option to specify from a drop-down list the underlying fine-tuned model to be used as well as choose between the two temperature settings (i.e., a standard mode with a temperature of 0.0 and a creative one with 0.7). This small-scale analysis already revealed a possible effect of position bias [25], where the position of the utilized model or hyperparameter has substantial influence on the users' decision to use it. That is, although the participants could pick the underlying LLM from a drop-down list, almost 90% of the time the participants left the fine-tuned GPT-3.5 Turbo, which was displayed as default first in the list, to be used. A similar observation can be made for the choice of temperature as 87% of the time, the choice was left to the pre-selected temperature of 0.0. If an error happened (i.e., the model hallucinated), the participants would rather repeat the same query or rephrase it, instead of changing the underlying LLM. This suggests that the burden of choice should be alleviated and an additional online model selection algorithm should be employed like the recently proposed time-increasing bandit algorithm from [26].

8. Conclusion

In this study, we explored the utilization of Large Language Models (LLMs) as intent-based copilots in modern visual-based chatbot building platforms. We demonstrated that fine-tuned LLMs outperform


Create dialog using Answers CoPilot

Explain what you want this dialog to do or what it needs to contain and Answers CoPilot will create it for you. Learn more

Name your dialog (optional)

Feedback dialog

Describe the dialog 0/160

Collect feedback from users about their experience with a product or service, prompting them to rate and provide comments.

Select mode of response

Standard

Expect to see common chat design patterns that are proven to work. You can rely on this mode to always generate consistent results.

Model

GPT-3

Figure 3: Visual interface (pop-up) used in our small-scale qualitative analysis that calls fine-tuned LLMs to generate a dialog based on the provided input text.

traditional prompt-based strategies with respect to accuracy and hallucinations. Interestingly enough, we found that a relatively tiny fine-tuned LLM can exhibit a comparable accuracy performance to the much larger GPT-3.5 Turbo while producing the least amount of hallucinations. Additionally, our research highlighted that more sophisticated prompting techniques like Chain-of-Thought not only improve the accuracy and lower the amount of hallucinations in a copilot setting, but can also lead to a lower bias toward popularity.

Limitations and Future Work. A limitation of our work is that we use a proprietary dataset for our experiments. We are aware that this does not foster reproducibility and plan in future work to open-source an anonymized dataset as these are yet to be prevalent in the research community for the task of creating copilots for visual chatbot generation. In addition, the research on LLMs is gaining a lot of traction and new models are being released with an increased frequency. Newer commercial models such as GPT-4o and Gemini 1.5 Pro have replaced their older variants and offer an improved performance. Although our study reports on models that were state-of-the-art at the time of research, we plan to continue our work by assessing newer and more sophisticated comercial and open-source model versions, but also look into sparsity techniques to reduce the number of active parameters in LLMs without ending up with a substantial loss in performance. Finally, we plan to conduct a longer online study to investigate the user experience and acceptance of intent-based copilots as well as explore the impact of different bandit techniques for forming an effective model selection.

References

[1] N. Sabharwal, A. Agrawal, N. Sabharwal, A. Agrawal, Introduction to google dialogflow, Cognitive virtual assistants using google dialogflow: develop complex cognitive bots using the google dialogflow platform (2020) 13–54.

[2] C. Bird, D. Ford, T. Zimmermann, N. Forsgren, E. Kalliamvakou, T. Lowdermilk, I. Gazit, Taking flight with copilot: Early insights and opportunities of ai-powered pair-programming tools, Queue 20 (2022) 35–57.

[3] C. Parnin, G. Soares, R. Pandita, S. Gulwani, J. Rich, A. Z. Henley, Building your own product copilot: Challenges, opportunities, and needs, arXiv preprint arXiv:2312.14231 (2023).

[4] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin, Attention is all you need, Advances in Neural Information Processing Systems 30 (2017).

[5] R. OpenAI, Gpt-4 technical report. arxiv 2303.08774, View in Article 2 (2023).

[6] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth, et al., Gemini: a family of highly capable multimodal models, arXiv preprint arXiv:2312.11805 (2023).

[7] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al., Mistral 7b, arXiv preprint arXiv:2310.06825 (2023).

[8] V. Murali, C. Maddila, I. Ahmad, M. Bolin, D. Cheng, N. Ghorbani, R. Fernandez, N. Nagappan, Codecompose: A large-scale industrial deployment of ai-assisted code authoring, arXiv preprint arXiv:2305.12050 (2023).

[9] P. Vaithilingam, E. L. Glassman, P. Groenwegen, S. Gulwani, A. Z. Henley, R. Malpani, D. Pugh, A. Radhakrishna, G. Soares, J. Wang, et al., Towards more effective ai-assisted programming: A systematic design exploration to improve visual studio intelli-code's user experience, in: Proc. of ICSE-SEIP'23, 2023.

[10] B. Tilekbay, S. Yang, M. A. Lewkowicz, A. Suryapranata, J. Kim, Expressedit: Video editing with natural language and sketching, in: Proc. of ACM IUI'24, 2024.

[11] B. Wang, Y. Li, Z. Lv, H. Xia, Y. Xu, R. Sodhi, Lave: Llm-powered agent assistance and language augmentation for video editing, in: Proc. of IUI'24, 2024.

[12] J. Prather, B. N. Reeves, P. Denny, B. A. Becker, J. Leinonen, A. Luxton-Reilly, G. Powell, J. Finnie-Ansley, E. A. Santos, "it's weird that it knows what i want": Usability and interactions with copilot for novice programmers, ACM Transactions on Computer-Human Interaction 31 (2023) 1–31.

[13] P. Liu, W. Yuan, J. Fu, Z. Jiang, H. Hayashi, G. Neubig, Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing, ACM Computing Surveys 55 (2023).

[14] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.

[15] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, Advances in NeurIPS (2022).

[16] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, et al., Scaling instruction-finetuned language models, Journal of ML Research (2024).

[17] T. Kojima, S. S. Gu, M. Reid, Y. Matsuo, Y. Iwasawa, Large language models are zero-shot reasoners, Advances in neural information processing systems 35 (2022) 22199–22213.

[18] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, W. Chen, Lora: Low-rank adaptation of large language models, In Proceedings of ICLR'22 (2022).

[19] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al., Llama: Open and efficient foundation language models, arXiv preprint arXiv:2302.13971 (2023).

[20] M. Xia, T. Gao, Z. Zeng, D. Chen, Sheared llama: Accelerating language model pre-training via structured pruning, in: Proc. of ICLR'24, 2024.

[21] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale, et al., Llama 2: Open foundation and fine-tuned chat models, arXiv preprint arXiv:2307.09288 (2023).

[22] K. Järvelin, S. L. Price, L. M. Delcambre, M. L. Nielsen, Discounted cumulated gain based evaluation of multiple-query ir sessions, in: Proceedings of ECIR'2008, Springer, Springer, 2008, pp. 4–15.

[23] G. L. Ciampaglia, A. Nematzadeh, F. Menczer, A. Flammini, How algorithmic popularity bias hinders or promotes quality, Scientific reports 8 (2018) 15951.

[24] H. Abdollahpouri, M. Mansoury, R. Burke, B. Mobasher, E. Malthouse, User-centered evaluation of popularity bias in recommender systems, in: Proc. of ACM UMAP'21, 2021.

[25] T. Joachims, L. Granka, B. Pan, H. Hembrooke, F. Radlinski, G. Gay, Evaluating the accuracy of implicit feedback from clicks and query reformulations in web search, ACM TOIS'07 25 (2007).

[26] Y. Xia, F. Kong, T. Yu, L. Guo, R. A. Rossi, S. Kim, S. Li, Which llm to play? convergence-aware online model selection with time-increasing bandits, in: Proc. of ACM Web Conference, 2024.