Research Article

Florian Lehmann* and Daniel Buschek

Examining Autocompletion as a Basic Concept for Interaction with Generative AI

Abstract: Autocompletion is an approach that extends and continues partial user input. We propose to interpret autocompletion as a basic interaction concept in human-AI interaction. We first describe the concept of autocompletion and dissect its user interface and interaction elements, using the well-established textual autocompletion in search engines as an example. We then highlight how these elements reoccur in other application domains, such as code completion, GUI sketching, and layouting. This comparison and transfer highlights an inherent role of such intelligent systems to extend and complete user input, in particular useful for designing interactions with and for generative AI. We reflect on and discuss our conceptual analysis of autocompletion to provide inspiration and a conceptual lens on current challenges in designing for human-AI interaction.

Keywords: autocompletion, interaction patterns, human-AI interaction, user-centred AI

1 Introduction

Autocompletion is a well established key feature in many applications today. It is most commonly used in search engines, such as Google, Bing, and Elasticsearch, by millions of users every day. For instance, when a user types a request into a web search, the system creates certain extended variations of the input and serves these back to the user. The autocompleted variations represent a list of search term suggestions. Then the user is free to choose from these autocompleted suggestions. The selection can be confirmed or edited further. In the context of search engines, such textual autocompletion is often called query autocompletion (QAC). In the context of software engineering, it is commonly referred to as autocomplete.

This approach is used in search engines to assist a user in formulating a proper input. Moreover, it is used in web browsers when typing a domain name, in code editors or IDEs when writing source code, and in other software tools where it is important to support the user in finding a precise input. Autocompletion can be considered a supportive technology to declare an input. As a side effect, autocompletion helps to make input faster [21]. However, the underlying key concept remains: Autocompletion continues and extends (partial) user input.

In light of this vital concept, we interpret autocompletion as a generative approach embedded in a user interface. When speaking of generative in the context of machine learning within this paper, we refer to those approaches that can be used to generate things. Methods to provide such generations can be found in the field of machine learning as well. Despite classifying input to make predictions, machine learning can also be used to generate data. Such generative models typically learn an underlying data distribution from which to sample new outputs. For example, generative machine learning approaches can complete pictures [31, 41, 46] and gestures [7], or extend texts [8, 35, 40]. Implementing generative machine learning into interactive software tools can open new possibilities. For instance, it can be used to transform digital sketches into mock-ups [32], create sketches from text descriptions [20], or to create digital wireframes from paper based sketches [9]. Another real world example is Kite,[1] a coding assistant based on the GPT language model.[2] It can generate complete methods from just a signature of methods, and a short, descriptive comment.

Although the latter examples rely on machine learning models, it is not mandatory to do so for an application to provide a generative feature. For instance, in the case of query autocompletion, it could be built on n-gram frequency statistics [29], and layout generation could be based on integer programming [14]. In particular, when analysing generative approaches from a perspective of

*Corresponding author: Florian Lehmann, Research Group HCI + AI, Department of Computer Science, University of Bayreuth, Bayreuth, Germany, e-mail: florian.lehmann@uni-bayreuth.de
Daniel Buschek, Research Group HCI + AI, Department of Computer Science, University of Bayreuth, Bayreuth, Germany, e-mail: daniel.buschek@uni-bayreuth.de

1 Kite: https://www.kite.com
2 Better Language Models and Their Implications (OpenAI GPT2): https://openai.com/blog/better-language-models

user interaction, it is hard to distinguish whether the application has machine learning implemented or not. The user interface functions as an abstract layer and hides the technology in the background. Accordingly, in this paper we regard "intelligence" as the ability to generate extended and ranked output based on partial user input, regardless of how this is technically achieved.

We assume that in the future, more and more applications will incorporate intelligent features. However, recent work by Yang et al. [44] pointed out challenges in designing applications with human-AI interaction. These include, for instance, the challenge of envisioning interaction with AI, in understanding AI capabilities, and in crafting interactions for unpredictable output.

Such challenges motivate us to reflect on – and learn from – existing interaction solutions as one approach towards informing future designs: In particular, in this paper, we revisit autocompletion as a reoccurring and reusable interaction concept for designing interaction with generative intelligent systems.

We contribute a conceptual analysis and transfer in three steps: First, we systematically analyse the underlying interaction and user interface of textual autocompletion, and extract its key conceptual elements.

Second, we identify these elements of textual autocompletion in other domains that use generative approaches, highlighting opportunities to transfer this concept and reuse it. Third, we reflect on potential benefits of this transfer in the light of the challenges of designing for human-AI interaction, and point out opportunities and challenges for future work.

2 Related Work

This research originated in the context of a broad literature review on topics combining HCI and AI. Within this research process we discovered similarities between autocompletion and the capabilities of generative machine learning approaches. In the following paragraphs, we summarise relevant work related to these topics.

2.1 Research on Autocompletion

Autocompletion is a broad topic with different research directions. A survey by Cai and Rijke [11] helps to gain a first overview of the most important topics: They indicate that most papers concentrate on the technical issues rather than on the user interface or interaction.

2.1.1 Frontend: User Interaction

Early HCI research on user interactions on textual autocompletion was done in 1986 by Jakobsson [21]. In Jakobsson's work, autocompletion was investigated as part of a library information system. The evaluation showed that textual autocompletion is more efficient to find entries in an information system in comparison to using shortcodes and a code catalogue. Besides research on efficiency, other work concentrated on engagement with the completed suggestions, for instance, how the input technique and suggestion ranking influences the selections by the users: Work by Mitra et al. [30] observed that users are more likely to engage with the autocompleted suggestions if the fingers have to travel longer between keystrokes or at word boundaries. Moreover, they showed that top ranked suggestions were preferred. A strong position bias was also found by Hofmann et al. [17]. They used eye-tracking to investigate how ranking positions affect user interaction. In their study participants focused on top-ranked suggestions regardless of whether the list war randomised or not. Others investigated how the organisation of suggestions influences user interaction [2]. For this, they compared alphabetical ordering with categorical ordering and composite. Their findings showed group and composite organisation to improve efficiency. Additionally, they suggest how to design for different organisation strategies.

2.1.2 Backend: Ranking, Personalisation, Modelling

Ranking and personalisation is a major theme in research on autocompletion with search engines. Models for improving ranking suggestions are also of importance. Thus, the core of research on backend functionalities for autocompletion concentrates on algorithms. As a part of that, research introduced an indexing data structure to improve the performance of query processing [5]. Focusing on adding context-sensitivity to algorithms, work by Bar-Yossef and Kraus [4] introduced and evaluated methods to incorporate users' search queries for suggestions. As well, they investigated how it affects ranking. Such context-sensitivity can be interpreted as personalisation of suggestion results. Adding personalisation to algorithms, research involved user-specific and demographic features [36]. Selective personalisation was investigated by Cai and de Rijke [10]. They showed that the typed prefix can indicate when it is appropriate to display personalised suggestion rankings. In another work [12], they introduced an approach to diversify the suggestion results. They aimed

to rank the intended term as high as possible while reducing redundancy in the list. For this, they evaluated a model that relies not only on current search popularity but also on within-session context. Including time-series in a model showed to further improve suggestion quality [37]. A comparison of eleven ranking approaches can be found in work by Di Santo et al. [15].

User interactions offer further possibilities to model autocompletion. For example, Li et al. [25] introduced a two-dimensional click model to better explain the vertical position bias and horizontal skipping bias. Incorporating the skipping behaviour into existing models they were able to improve efficiency. Predictions on the search intent can be done based on keystrokes and clicks [24]. Furthermore, interactions with apps can be used to rank suggestions in search on mobile devices [48]. Instead of such active user feedback, also implicit negative feedback, for instance skipping suggestions or dwell time can be involved to model suggestions. In particular, Zhang et al. [47] used dwell time and position of unselected suggestions as features for implicit negative feedback to adapt the ranking of query suggestions.

2.2 Autocompletion Is not Only for Text

Beyond text, autocompletion can be applied to an array of other domains, for instance, sketching, image editing, animation, and many more. It was integrated into a GUI-based tool to create XML [27]. Bennett et al. [7] presented approaches for gestural autocompletion. They showed that autocompletion improves gestures: These were shorter, more accurate, and faster to execute. Others focused on sketches and reported on a system to autocomplete digitally sketched symbols [13, 39]. Further work proposed a framework to enable autocompletion on value cells in relational tables such as spreadsheets [49]. In the context of code, Pythia [38] offers code completion based on a neural net to rank method and API suggestions. Work on creative domains demonstrated that an RNN, trained on physics-based simulations, can be used to autocomplete keyframe animations [50]. Also, Hsu et al. [19] presented autocompletion for aggregate elements that can be used for 2D planes, 3D surfaces, and 3D volumes. Feedback on their approach showed that workload can be reduced and the system encouraged participants to explore more variations.

2.3 Generative Machine Learning Has Autocompletion Capabilities

Within our literature research, we observed opportunities for connecting autocompletion as a concept with generative machine learning, and vice-versa. In this light we highlight some generative machine learning approaches in the following. In particular they share the capability to make partial user input more complete. One example for that is work by Park and Chiba [33], using neural networks for textual autocompletion.

In general, generative approaches in machine learning are used to generate data based on prior observations, and are thus different, for example, from classification tasks. Generation has gained increasing attention with the rise of Deep Learning: For instance, generative approaches can be used to model language for various NLP tasks. Work by Vaswani et al. [40] introduced Transformer networks. A Transformer relies solely on self-attention, this replaced recurrent layers which were commonly used with encoder-decoder architectures in the past. The language model GPT-3 is based on a Transformer architecture [8], which is a further developed version of GPT-2 [35]. GPT-3 is considered a state-of-the-art model that can fulfill different functions: For example, it can generate news articles, translations, or correct English grammar.

Such machine learning models can also be integrated into creative applications: For example, Huang and Canny [20] introduced a system to generate sketches from text input. Another paper introduced AI tools that can help to support UI designers [32]. One of their tools can detect UI elements on low-fidelity sketches, which then can be transformed into a medium fidelity mock-up. Other work presented a tool to transform analog sketches into digital wireframes [9].

However, it is not always a must for intelligent, interactive applications to rely on machine learning. In a comparison within this paper, we involved, for instance, a tool that is not based on machine learning, however, it can be used to automatically solve (and thus generate/complete) layouts for user interfaces [14]. In this paper, we follow a user-centred view [43, 44]: In particular, we aim to take a conceptual yet concrete step on the path towards user-centred, interactive AI by examining an existing interactive concept – autocompletion – in the "new" light of generative computational capabilities.

3 The Concept of Autocompletion

Our analysis starts by describing autocompletion on a conceptual level. Thereafter, we go more into detail until we have formally understood what autocompletion is. For describing autocompletion, we refer to the case of textual autocompletion since this is currently its most common application.

3.1 Overview and Delineation

On a conceptual level, autocompletion has the role to continue, extend or complete digital content. This could be any input made by a user. More specifically, such input is processed by a system in order to generate an extended version. For example, in probabilistic terms, the model samples continuations conditioned on the user input. These different variations of the extended input are then presented to the user. The user is then able to select one of the suggestions or ignore them. Either the completion was successful, or the user decides to further specify the original intent. Autocompletion allows for close-loop interaction where the system reacts to the user and vice versa.

As a software feature, autocompletion (also: autocomplete) is used in search engines to formalise a query, in content management systems to complete category names, and on smartphones to predict the next word.

Similar software features are auto fill and auto correct [3]. Auto correct is sometimes implemented together with autocompletion. This supports the user to correct the faulty input and then suggests an autocompletion based on the corrected input. This might increase the overall convenience for the user when working with textual input. Auto fill instead aims to complete form input. A common technique is to detect the form field identifiers and recognise past input. If there was past input on similarly named input fields, then the software will suggest to complete the form automatically. Compared to auto correct, auto fill might less often appear together with autocomplete.

3.2 Technical Approaches

Here we outline some technical approaches that can be used to enable a computational system for autocompletion capabilities.

3.2.1 Approaches in Industry and Commercial Products

In commercial products, transparency on how systems manipulate the input to complete it, is typically not offered to the end-user. Algorithms are kept secret and the user interface works as an abstract layer for the user to keep interaction simple and hide the technical functions.

Insights from a practical perspective on how textual autocompletion works, however, can be found in the elasticsearch documentation. Elasticsearch is an open source system that provides autocompletion capabilities out of the box. Its documentation describes how n-gram frequency statistics enable autocompletion.[3]

3.2.2 N-gram Frequency (non Machine Learning)

In textual autocompletion n-gram frequency statistics are commonly applied. N-grams are substrings of a string with a length of n. For example: "Hello World" split in n-grams of length three, with a sliding window, would result in "Hel", "ell", "llo", "lo", and so on. Strategies might differ here, e. g. whitespaces could be removed first. Frequency statistics are obtained by counting the appearance of the n-gram across all known n-grams of search terms in the database. The frequency can be used to determine a likelihood to pick a certain search term from the database. Only search terms with a high likelihood are going to be returned as suggestions to the user. There is also already existing work on this topic [29].

3.2.3 Machine Learning

Machine learning and neural networks, are a broad topic. Here, we will only mention which architectures can be used for generating data. Moreover, we highlight those approaches that can be utilised to extend partial text or image input.

In the field of machine learning there can be found various methods to generate text. In some cases, such approaches still need to generate n-grams first. Instead of just relying on frequency statistics, the n-grams are used to train neural networks. These neural networks are then used to generate completed versions of the partial user input. As well, there exist models that do not need to split

---

3 Elasticsearch Suggesters based on n-grams: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-suggesters.html

words into n-grams at all, for instance the continuous bag of words model (CBOW) and skip-gram [28]. Work by Park and Chiba [33] utilised a neural net with along short-term memory (LSTM) architecture to generate the next word of a query. The performance of a neural net depends on input data and architecture. LSTMs and recurrent neural networks (RNNs) in general, turned out to work well with text data [16]. However, latest advances in the field found Transformer networks to be superior for textbased tasks [8, 35, 40]. Models vary across domains. For instance, images can be automatically inpainted by utilising convolutional neural networks (CNNs) [31, 41, 46]. Moreover, recent progress in image generation often uses Generative Adversarial Networks (GANs) (e. g. [22]).

4 Comparing Textual Autocompletion with Generative Approaches

We have described autocompletion on a conceptual level and gave an introduction to technical approaches. Next, we dissect the user interface and interaction patterns of a practical example, namely textual autocompletion for search queries. This is followed by comparing and connecting textual autocompletion to generative approaches. These generative approaches are part of applications from related work or real world applications: In particular, we examine code completion,[4] sketch completion (e. g. [32]), and layouting (e. g. [14]) to textual autocompletion.

More specifically, we first provide background information about the examples. Then we compare the user interface and finally the interaction. Parts of this comparison are similar to our prior work [23].

4.1 Textual Autocompletion in Search Engines (Query Autocompletion)

User Interface: We created a wireframe of an example of a textual autocompletion in current search engines as seen in Figure 1 on the left. It is a composition of three input elements. The visual design can be adapted in a way that the elements are merged and look like a single, reactive element, as seen in Figure 1 on the right. Yet, the basic elements remain the same. In summary, textual autocompletion consists of interface elements as follows:
– Input field for text (input)
– List to display completed suggestions (suggestion area)
– Button to confirm final input (button)

Those elements can be varied across implementations. For example, the list can be displayed horizontally, instead of vertically, and it must not be a list at all. The suggestion area can be any other element as long as it is suitable to display an array of suggestions. As well, the confirm button could be hidden at the beginning and fades in after text was typed.

Interaction: The UI may be structured as seen in Figure 2, example one. We observed the following interactions and put them into a flowchart as seen in Figure 3, chart one: The interaction starts with selecting the input field. Then a letter is typed with the keyboard. Next, the user can choose to confirm the input or select one of the suggestions. After selecting a suggestion, the user can again choose to confirm the input or select another suggestion. Alternatively, the text can be further edited with the keyboard by typing another letter or correcting prior input. The interaction finally ends with a confirmation.

---

4 PyCharm code completion: https://www.jetbrains.com/help/pycharm/auto-completing-code.html

Figure 1: The figure shows the wireframes of textual autocompletion in the special case of query autocompletion. Both versions offer the same functionality. They make partial user input more complete. On the left: A user interface composed of three components, an input field, a suggestion area, and a button for confirmation. With loose interface elements. On the right: A more compact version with adapted visual composition. The search button was replaced by an icon of a magnifying glass. The search field and the suggestion area are combined into one field. Both examples show an option in the bottom right of the suggestion area to report inappropriate entries.
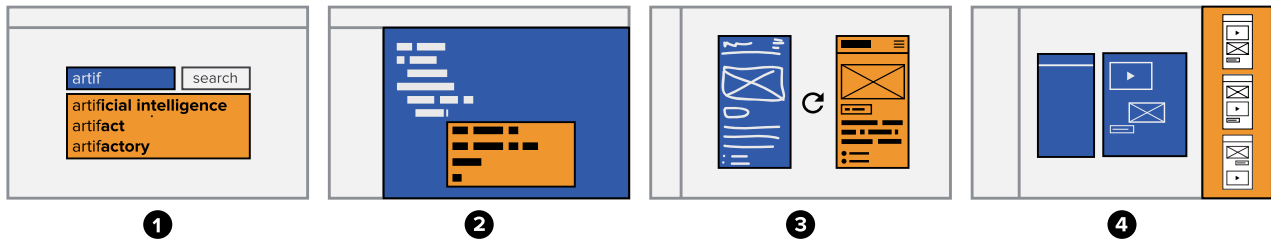
Figure 2: Comparison of the user interface of 1) search query autocompletion and three examples that share the same underlying interaction concepts, namely 2) code completion, 3) mock-up generation from sketches, and 4) layout solving. Coloured areas highlight similarities in the user interface. Blue areas with white symbols indicate fields for user input, orange areas with dark grey symbols indicate fields for completed input by the AI. Example three is inspired by [32], Example four is inspired by [14]. All examples have the main function to make something (partial user input) more complete.
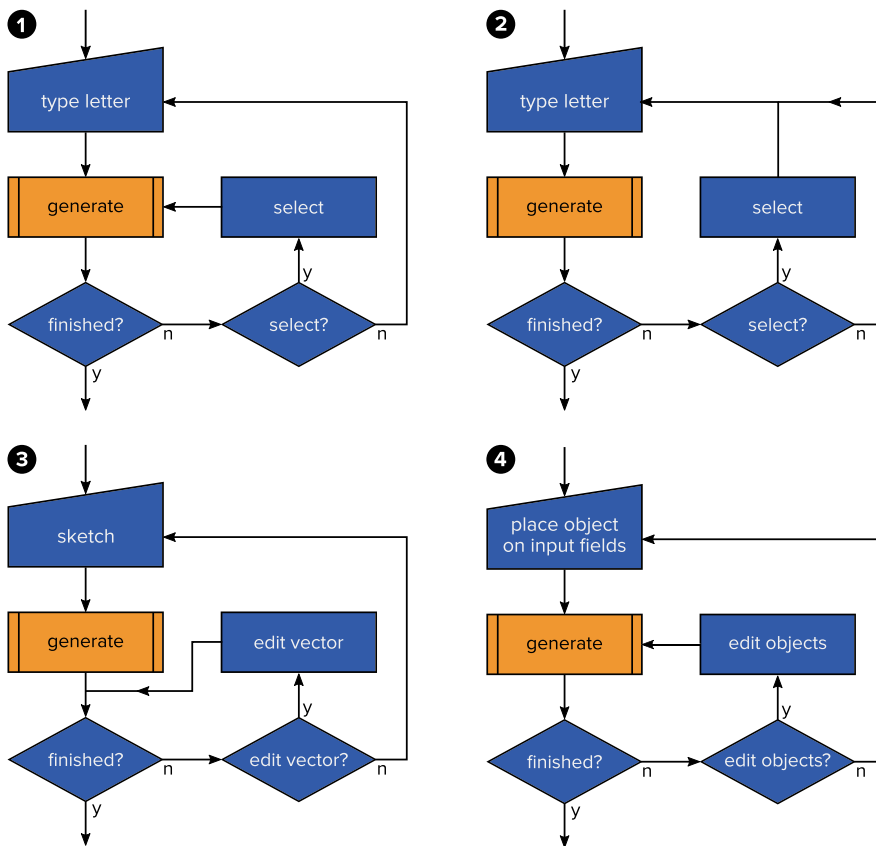

Figure 3: Visualisation of interaction patterns as flow charts. Presenting 1) search query autocompletion and generative approaches, such as 2) code completion, 3) mock-up generation from sketches, and 4) layout solving. Blue elements indicate user interaction, orange areas indicate system interaction. Chart three relates to [32], chart four relates to [14]. Even though details in the interaction loops differ slightly, they all share the similarity to have a generative intelligent system in the loop. The generative process aims to make partial user input more complete. The final decision to accept a generated object is made by the user.

If a user finds a suggested entry inappropriate it can be reported through a link in the bottom right of the suggestion area.

The interaction flow is quite simple. Yet, it offers functionality that is widely accepted for instance in search engines. Because of its simplicity we find it promising to transfer this basic flow to applications that offer human-AI

interaction. As well, it holds important information: It allows for continuous interaction between the system (backend) and the user (through the frontend). Moreover, the final control is left to the user. For more complex tasks this interaction flow might have to be extended. Here, suggestions could involve all sorts of contextual data, for instance location, intent, or emotion. Output by the systems could

be communicated to the user differently, for instance by an agent through conversational approaches.

4.2 Code Competion in Code Editors / IDEs

We consider code to be a formal and structured type of text. It offers informal and structural benefits over plain text since specific parts have special functions. For example, a word could be a method, variable, or class. This is used by modern IDEs to enable for more convenient work with code. One of modern IDEs core features is code completion which is close to well-known textual autocompletion. But services like TabNine[5] and Kite[6] offer more intelligent features, since they rely on neural nets. This is also part of current research [38] to improve the code suggestions.

User Interface: A generalised user interface for autocompletion in a code editor can be seen in Figure 2, example two. The user enters code into a text area (blue). Suggestions appear in a pop-up widget (orange). The widget is placed below the cursor. The widget's left edge aligns with the input cursor. The widget displays an ordered list of suggestions, e. g. method signatures. It is only visible after typing. A selected suggestion appears at the position of the input cursor.

Interaction: The interaction for autocompletion in code editors or IDEs is similar to textual autocompletion in search engines. The interaction flow for autocompletion is depicted in Figure 3, chart two. The interaction starts after the cursor is set in the textarea. The users enters input via the keyboard. This input is used to generate suggestions. If the input is not finished, the user can decide to select a suggestion, or ignore it. On selection, the suggested code is placed at the position of the cursor. Now, the loop starts over again.

4.3 Intelligent UI Sketching Tools

Contrary to code completion, image generation systems are not part of nowadays workflows in image and GUI editing tools. The latest efforts in research, however, show progress in this area. For instance, completion of partially drawn sketches [39] and transforming paper drawn sketches to digital wireframes [9]. Similar to the latter, another tool can generate medium fidelity mock-ups from low fidelity sketches [32].

---

5 TabNine: https://www.tabnine.com
6 Kite: https://www.kite.com

User Interface: For the description of the user interface, we orient on already existing work [32], and depicted the UI in Figure 2, example three. The user sketches on a canvas element (blue). In fixed intervals, a medium fidelity mock-up appears on another canvas element on the right hand-side (orange). Between the two canvas elements, a button is displayed to manually trigger the generation of the medium fidelity mock-up.

Interaction: The user starts sketching with a digital brush or pen tool on a canvas element. That is when the interaction starts, see Figure 3, chart three. The system generates in fixed intervals a medium fidelity mock-up consisting of vector graphics. The user can then accept the mock-up by saving it. Alternatively, the user can edit the vector elements and save the mock-up afterwards. The user could also modify the sketch to modify the mock-up.

4.4 Layout Generators

Similar to code completion for textual autocompletion, solving layouts can be considered a specific problem within the domain of graphical user interfaces. To arrange a user interface, layout possibilities increase with the number of interface elements. There are logical constraints, however, that limit the variations. For final results, some variants are to be preferred over others. Layouting itself is a time consuming manual task. Recently, an interactive layout solving tool was introduced, on which we orient for dissecting the user interface and interaction [14].

User Interface: A generalised wireframe of the user interface can be seen in Figure 2, example four. Pre-defined interface elements are presented in a toolbar. These elements can be dragged and placed into a workspace area (the blue area on the right-hand side). Here, the elements are placed without depending contextually on each other. On the left is another workspace area (the more narrow, blue area). However, in that area, elements are arranged to constrain the layout. A toolbar on the right (orange) displays all suggested layout solutions.

Interaction: The interaction starts when the user places objects on the workspaces. Compare Figure 3, chart four. After all objects have been placed, the generation is triggered by the user manually, or in fixed intervals. Subsequently, the layout solver combines the inputs and generates layout variations. The user is free to save layout suggestions and finish interaction. If the layouts are not sufficient, the user can decide to edit the objects to generate new layouts. Alternatively, the user can place new objects to start over with the interaction process.

4.5 Comparing the UI of Generative Approaches with Query Autocompletion

For the comparison of the user interface, we have visualised simplified wireframes, as seen in Figure 2. These wireframes help us to highlight the important parts of the user interface. The blue regions indicate fields for user input. The orange fields indicate the area for system generated suggestions. We observed similarities between all interfaces. Even though layouts differ from example to example, the function remains the same. All applications provide a field for input. Here, only partial and incomplete input is done by the user. Then the system generates an extended version of it. Where possible, the system generates a set of distinct suggestions for output. The only exception is example three in Figure 2, since the system outputs only one suggestion. All examples, however, share an area where the generated output is presented to the user. These areas are aligned near the input to allow the user to parse the generated output easily. Moreover, all applications use the input and output fields as the primary elements for user interaction.

4.6 Comparing the Interaction of Generative Approaches with Query Autocompletion

Similar to the comparison of the user interface, we have also dissected the user interactions for each example and visualised this in Figure 3. We relied on flow charts to visualise the interaction in a formal style. Blue elements indicate user interaction, orange elements indicate system operation. For our interaction flow charts, we oriented on practical examples such as Google Search, Jetbrains PyCharm IDE, and examples from related work [14, 32]. As seen in Figure 3, the interaction flows are all similar. However, they differ slightly between examples, chart one and chart two in Figure 3 are good examples for these minimal differences in interaction: After selecting a suggestion in chart one, the system instantly generates another suggestion. This is different to chart two where the user needs to provide manual input before the next suggestion is generated. These are slight differences, however, they can influence the workflow crucially since both flows are very repetitive and happen very often when a user executes input. We want to note, that our flow charts are abstract and generalised – specific implementations use altered interaction flows. Our visualisation however, highlights that all approaches have a generative intelligent system in the loop. The user interaction starts with the operation found at the top of each flow chart. Followed by the generative process

operated by the intelligent system in the backend. Afterwards, the user has to decide to accept the result. If not, the user can select a suggestion, further refine it, or ignore it and feed new partial input to the generative process. The "finish" switch is symbolic and indicates an end of the user interaction. This could be finalising the input, e. g. by running a search query, starting a new line in a code editor, saving a wireframe, or saving a layout. All examples are user-centred since the user is always in control and can decide whether to accept a generated object or not. In general, the role of the intelligent system is to make the user input more complete.

4.7 Aspects of Query Autocompletion

Based on our analysis of user interfaces and interaction flows across different domains, as seen in Figure 2 and Figure 3, we derived five aspects that define autocompletion from a user-centred perspective. These aspects can help to inform and design novel applications which allow for human-AI interaction. We summarise and highlight these aspects additionally in Table 1. In the following, we describe them in more detail:

User Interface – The user interface serves as an abstract visual layer that reveals the functions to the user at the frontend. This way it is rather less important for the user to understand the underlying technology in the backend. A minimalistic user interface suitable for autocompletion should hold a field for input and a suggestion area. Generated objects should be placed near, but separate from the input.

Workflow – The user interfaces allow for continuous interaction between the system and the user. The system generates suggestions interactively and can become also a part of the workflow. The interaction between the system and the user is continuous until the input is finally confirmed.

User Decision – The user can freely decide to accept a suggestion or not. The suggestions can be ignored which underlines the supportive and rather passive role of the system. In case that the user ignores the suggestions, the system keeps on generating new versions.

Editing – The user can freely edit a taken suggestion until it fits the intent. The user can extend the suggestions, or delete them. The system should be tolerant of errors, by automatically suggesting corrections or informing the user about an error.

Information – The user input is never considered complete, always partial input is served to the system. The underlying AI (backend) is conditioned to predict a more

Table 1: We identified five inherent aspects of autocompletion in interactive applications. Applications offering autocompletion help to make partial user input more complete. We propose that these five aspects could be transferred to other interactive AI applications that offer features to extend partial user input.

Aspect	Description
User Interface	The interface holds a field for input. Generated objects are placed near the input.
Workflow	One or more suggestions are generated interactively and are part of the workflow.
User Decision	The user can decide to accept a suggestion or not.
Editing	The user can further edit the suggestion object.
Information	Partial user input serves as input for the system. The AI's prediction is conditioned on the input to extend it.

complete version of this partial user input. Information can not only be retrieved from the explicit input, but also from other implicit variables, such as dwell time.

5 Discussion

Here, we reflect on the described conceptual connections, drawn between autocompletion and AI with generative capabilities, for integration of intelligent features in today's applications.

5.1 Understanding the Concept of Autocompletion

With our analysis, we examined autocompletion on a conceptual level. In more detail, we looked at the user interface and the interaction but gave also brief descriptions of the underlying technology. Because it is well-known from search engines, we used text query autocompletion as an example for autocompletion. Yet, our analysis goes beyond textual autocompletion by considering other domains and different approaches, such as systems that rely on machine learning methods to generate wireframes from digital sketches.

We found autocompletion to share similarities in the user interfaces across domains, as well as a basic interaction flow. We identified five aspects inherent to autocompletion applications. With our comparison of query autocompletion with other generative approaches, we demonstrated that one role of intelligent generative systems is to extend and continue partial user input.

Our research here is conceptual: We did not run a user study or analyse empirical data. We rather analysed the state-of-the-art of today's applications which integrate autocompletion. We outlined five aspects that are common to applications completing user input: The user interface, the workflow, the user decision, editing, and information.

A summary can be found in Table 1. These aspects can provide a high level starting point for interaction with applications capable of generating things. We particularly expect these to prove useful for designing *computational* (AI) tools since they keep the focus on designing interactions, not interfaces [6].

As a key aspect of this work, we revealed analogies in graphical user interfaces and interaction flows between query (textual) autocompletion and other intelligent, generative approaches. In terms of the user interface, we looked for the function of an interface element and compared it between applications, as seen in Figure 2. For this, we disregarded the visual design. For real world deployments, this means that such interface elements may look completely different but still serve the same function. Different factors could influence the visual design, such as the type of data, alignment of the interface elements, input modalities, device constraints, and so on. We suggest interpreting the user interface as a rather abstract layer that simplifies working with the underlying technology. If we can understand the function of interface elements, then we can transfer them to different domains, as recognised for autocompletion here.

For the user interaction, we explicated them using flow charts, as seen in Figure 3. The interaction between the user and intelligent systems then can be understood as a sequence of actions over time. By visually coding the operations in the flow chart, we identified an interaction loop in all examples. However, the flow charts capture only one generalised interaction flow in each example. More variants might exist, depending on the use case and implementation. Still, we suppose the user-centred interaction flow to be persistent.

In general, the detailed inspection of autocompletion here demonstrates potential benefits and insights gained from analysing and dissecting patterns in already existing interfaces, interactions, and technologies.

5.2 Generative Systems Can Be Used to Extend and Complete User Input

We revealed an inherent role in intelligent generative systems by analysing autocompletion on a conceptual level and underlining its function in different application domains. This role is "to extend and complete user input". In the context of our work, we summarised different approaches as "intelligent" as long as they showed the capability to extend partial user input, and provide more completed suggestions to the user. For example, this could be statistical methods from NLP, integer programming, or machine learning. Similar to the examples for autocompletion we have examined in our analysis, such generative approaches from machine learning work with partial input and generate extended data. Thus, one role of generative machine learning is "to extend and complete user input", and it is not only limited to text data.

Looking ahead, for example, a neural net might generate a complete text document from only a few keywords. In a scenario like this, we could assign at least one more role to generative machine learning. For instance, the role "to inspire the user". This would make the role of the system more active. At the same time, we would have to assume a role change in the user. The user would rather become an editor, instead of being an author. Besides text, this might also be anticipated for other domains, for example user interface design. Generative systems are able to generate functional wireframes from paper sketches [9]. Taking this idea to a scenario where the user provides only sketches and the system returns complete visual designs, we assume the roles would change similarly to text generation.

Systems that merely provide autocompletion, however, are less active, more passive. We consider them to be user-centred, since the user is in control of the system. The system output depends on the partial user input. Furthermore, the user is free to decide to accept a suggestion or to ignore it.

Considering the progress of machine learning over the last years it might be possible that future machine learning models will increase in performance. Their capabilities will improve, as well, tasks will be more complex. Given the techniques and computational power is evolving as in the recent past. This might open new possibilities and thus it is likely that intelligent features will be implemented in applications. We suggest to rethink the role of such systems in general, and how we want to integrate them into our workflows in the future.

5.3 Addressing Challenges in Human-AI Interaction

As alluded to in the intro, it is a recognised challenge to design for interactive applications of AI and recent work [44] has summarised these design challenges. Here we examine three of them to reflect on and discuss in the light of our work in this paper.

5.3.1 The Challenge of Envisioning Interaction with AI

Similar to our investigation of autocompletion in this paper, we suggest to analyse already existing intelligent tools for inspiration. Ideally, these can be dissected into reusable interface and interaction patterns, as shown in our example here. Developing such a set of interface and interaction patterns over time might then facilitate composing new interactions with intelligent systems.

5.3.2 The Challenge of Understanding AI Capabilities

The autocompletion pattern might also facilitate understanding of AI, concretely, by putting input-output mappings at the core of the interaction, thus making them *explorable*. First, the AI is fed with partial user input, which the user can quickly vary and iterate on to explore "AI reactions" and thus potentially develop a (tacit) understanding of it, possibly similar to experiences with rule-based systems. Second, autocompletion typically generates *multiple* variants as output. This might help the user to judge AI capabilities, since it gives a glimpse at the AI's potential output space, especially also across repeated input variation/iteration. Third, autocompletion typically *ranks* output, for example by probability, which might facilitate user understanding of AI capabilities as it gives a simple way of directly indicating the AI's (relative) uncertainty in the GUI.

5.3.3 The Challenge of Crafting Interactions for Unpredictable Output

The output of intelligent systems can be unpredictable from time to time. For instance, intelligent systems might generate text or images that are inappropriate to the user. Autocompletion provides an example UI for addressing this challenge in three ways: First, by design, it leaves the final decision about the accepted content to the user. Second, it shows multiple options, thus possibly including

not only an "outlier" but also more appropriate alternatives (or at least supporting the discovery of an "outlier" as such). Third, as depicted in Figure 1, the autocomplete UI easily affords to give explicit feedback on inappropriate output. This way, a personalised filter could be created over time or the signals could be used for the next training iteration of a machine learning model to keep out inappropriate output in the future.

In summary, our suggestions on the design challenges illustrate how a well-known UI/interaction concept such as autocompletion can be used as a conceptual lens and starting points to design for interactive intelligent systems.

For realising this potential in practice we see the key in collaborations between domain experts: This could help to integrate intelligent features in future workflows within applications. Moreover, this might simplify practical work with machine learning as a design material. This is important since machine learning is supposed to add complexity to software architecture and interfaces at the same time (e. g. Yang et al. [42]). This trade-off between functional complexity and a simplistic user interface should be addressed when designing for intelligent systems. The complexity should be reduced at least for the user of a human-AI application.

5.4 Mixed-Initiative User Interfaces for Human-AI Interaction

Beyond these design challenges, intelligent applications offer great potential, for example, to support finding ideas [1, 26, 45]. In general, we see the opportunity to connect the ideas of this work. For instance, to sense of agency. Especially for user-centred approaches, it is important to measure how much the user feels in control over a tool. Another aspect that deserves more attention is timing of AI capabilities in interactive use. For example, timing of updates in autocompletion are driven by the user (e. g. typing another character triggers updated completions). However, one might study when to offer autocompletion at all (e. g. for text completion), since it also requires attention (cf. [34]). This could be combined with research on negative feedback and error-tolerance. Negative feedback could be used to infer actions to adapt the AI involvement at the interface level accordingly.

Both timing and sense of agency could be examined in particular in light of the concept of mixed initiative interfaces [18]. Our analysis of autocompletion also already connects to this – both user and AI system contribute to the emerging digital content (e. g. query, text, image) via a specific input-generation-selection loop (Figure 3).

6 Conclusion

With this paper, we examined autocompletion on a conceptual level and analysed its interface, interaction, and technical elements. We identified reoccurring interface and interaction patterns in autocompletion across several domains, in particular going beyond the "traditional" text query completion. For example, we recognised analogies to autocompletion in AI-support for digital sketches and layouting. Based on our conceptual analysis, we suggested and discussed autocompletion as an inspiration and conceptual lens on current challenges in designing for human-AI interaction. With this work, we hope to provide a pragmatic, concrete conceptual starting point to help envision interaction designs with and for AI that can generate new things.

As future work, we plan to conduct experimental studies to empirically investigate the transfer and use of autocomplete UIs for interaction with generative AI as conceptually extracted here. More broadly, the highlighted inherent aspects of the autocomplete pattern further motivate investigations in combination with topics from mixed initiative interaction, sense of agency, and timing.

References

[1] Alberto Alvarez, Steve Dahlskog, Jose Font, Johan Holmberg, Chelsi Nolasco, and Axel Österman. Fostering creativity in the mixed-initiative evolutionary dungeon designer. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, pages 1–8, Malmö Sweden, August 2018. ACM. ISBN 978-1-4503-6571-0. 10.1145/3235765.3235815. URL https://dl.acm.org/doi/10.1145/3235765.3235815.

[2] Alia Amin, Michiel Hildebrand, Jacco van Ossenbruggen, Vanessa Evers, and Lynda Hardman. Organizing Suggestions in Autocompletion Interfaces. In Mohand Boughanem, Catherine Berrut, Josiane Mothe, and Chantal Soule-Dupuy, editors, *Advances in Information Retrieval*, volume 5478, pages 521–529. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. ISBN 978-3-642-00957-0 978-3-642-00958-7. 10.1007/978-3-642-00958-7_46. URL http://link.springer.com/10.1007/978-3-642-00958-7_46. Series Title: Lecture Notes in Computer Science.

[3] Nikola Banovic, Ticha Sethapakdi, Yasasvi Hari, Anind K. Dey, and Jennifer Mankoff. The Limits of Expert Text Entry Speed on Mobile Keyboards with Autocorrect. In *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 1–12,

Taipei Taiwan, October 2019. ACM. ISBN 978-1-4503-6825-4. 10.1145/3338286.3340126. URL https://dl.acm.org/doi/10.1145/3338286.3340126.

[4]   Ziv Bar-Yossef and Naama Kraus. Context-sensitive query auto-completion. In *Proceedings of the 20th international conference on World wide web*, WWW '11, pages 107–116, New York, NY, USA, March 2011. ACM Press. ISBN 978-1-4503-0632-4. 10.1145/1963405.1963424. URL https://doi.org/10.1145/1963405.1963424.

[5]   Holger Bast and Ingmar Weber. Type less, find more: fast autocompletion search with a succinct index. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '06, pages 364–371, New York, NY, USA, August 2006. ACM Press. ISBN 978-1-59593-369-0. 10.1145/1148170.1148234. URL https://doi.org/10.1145/1148170.1148234.

[6]   Michel Beaudouin-Lafon. Designing interaction, not interfaces. In *Proceedings of the working conference on Advanced visual interfaces – AVI '04*, page 15, New York, NY, USA, 2004. ACM Press. ISBN 978-1-58113-867-2. 10.1145/989863.989865. URL http://portal.acm.org/citation.cfm?doid=989863.989865.

[7]   Mike Bennett, Kevin McCarthy, Sile O'Modhrain, and Barry Smyth. SimpleFlow: Enhancing Gestural Interaction with Gesture Prediction, Abbreviation and Autocompletion. In Pedro Campos, Nicholas Graham, Joaquim Jorge, Nuno Nunes, Philippe Palanque, and Marco Winckler, editors, *Human-Computer Interaction – INTERACT 2011*, Lecture Notes in Computer Science, pages 591–608, Berlin, Heidelberg, 2011. Springer. ISBN 978-3-642-23774-4. 10.1007/978-3-642-23774-4_47.

[8]   Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language Models are Few-Shot Learners. *arXiv:2005.14165 [cs]*, July 2020. URL http://arxiv.org/abs/2005.14165. arXiv:2005.14165.

[9]   Daniel Buschek, Charlotte Anlauff, and Florian Lachner. Paper2Wire: a case study of user-centred development of machine learning tools for UX designers. In *Proceedings of the Conference on Mensch und Computer*, MuC '20, pages 33–41, New York, NY, USA, September 2020. Association for Computing Machinery. ISBN 978-1-4503-7540-5. 10.1145/3404983.3405506. URL https://doi.org/10.1145/3404983.3405506.

[10]   Fei Cai and Maarten de Rijke. Selectively Personalizing Query Auto-Completion. In *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, SIGIR '16, pages 993–996, New York, NY, USA, July 2016. ACM Press. ISBN 978-1-4503-4069-4. 10.1145/2911451.2914686. URL https://doi.org/10.1145/2911451.2914686.

[11]   Fei Cai and Maarten de Rijke. A Survey of Query Auto Completion in Information Retrieval. *Foundations and Trends® in Information Retrieval*, 10(4):273–363, September 2016. ISSN 1554-0669, 1554-0677. 10.1561/1500000055. URL

https://www.nowpublishers.com/article/Details/INR-055. Publisher: Now Publishers, Inc.

[12]   Fei Cai, Ridho Reinanda, and Maarten De Rijke. Diversifying Query Auto-Completion. *ACM Transactions on Information Systems*, 34(4):25:1–25:33, June 2016. ISSN 1046-8188. 10.1145/2910579. URL https://doi.org/10.1145/2910579.

[13]   Gennaro Costagliola, Mattia De Rosa, and Vittorio Fuccella. Investigating Human Performance in Hand-Drawn Symbol Autocompletion. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 279–284, October 2013. 10.1109/SMC.2013.54. ISSN: 1062-922X.

[14]   Niraj Ramesh Dayama, Kashyap Todi, Taru Saarelainen, and Antti Oulasvirta. GRIDS: Interactive Layout Design with Integer Programming. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, pages 1–13, New York, NY, USA, April 2020. ACM Press. ISBN 978-1-4503-6708-0. 10.1145/3313831.3376553. URL https://doi.org/10.1145/3313831.3376553.

[15]   Giovanni Di Santo, Richard McCreadie, Craig Macdonald, and Iadh Ounis. Comparing Approaches for Query Autocompletion. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 775–778, New York, NY, USA, August 2015. ACM Press. ISBN 978-1-4503-3621-5. 10.1145/2766462.2767829. URL https://doi.org/10.1145/2766462.2767829.

[16]   Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, November 1997. ISSN 0899-7667, 1530-888X. 10.1162/neco.1997.9.8.1735. URL http://www.mitpressjournals.org/doi/10.1162/neco.1997.9.8.1735.

[17]   Kajta Hofmann, Bhaskar Mitra, Filip Radlinski, and Milad Shokouhi. An Eye-tracking Study of User Interactions with Query Auto Completion. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, CIKM '14, pages 549–558, New York, NY, USA, November 2014. ACM Press. ISBN 978-1-4503-2598-1. 10.1145/2661829.2661922. URL https://doi.org/10.1145/2661829.2661922.

[18]   Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, CHI '99, pages 159–166, New York, NY, USA, May 1999. ACM Press. ISBN 978-0-201-48559-2. 10.1145/302979.303030. URL https://doi.org/10.1145/302979.303030.

[19]   Chen-Yuan Hsu, Li-Yi Wei, Lihua You, and Jian Jun Zhang. Autocomplete Element Fields. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pages 1–13, New York, NY, USA, April 2020. ACM Press. ISBN 978-1-4503-6708-0. 10.1145/3313831.3376248. URL https://dl.acm.org/doi/10.1145/3313831.3376248.

[20]   Forrest Huang and John F. Canny. Sketchforme: Composing Sketched Scenes from Text Descriptions for Interactive Applications. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pages 209–220, New York, NY, USA, 2019. ACM. ISBN 978-1-4503-6816-2. 10.1145/3332165.3347878. URL http://dl.acm.org/doi/10.1145/3332165.3347878.

[21]   M. Jakobsson. Autocompletion in full text transaction entry: a method for humanized input. *ACM SIGCHI Bulletin*, 17(4):327–332, April 1986. ISSN 0736-6906.

10.1145/22339.22391. URL https://doi.org/10.1145/22339.22391.

[22] Tero Karras, Samuli Laine, Miika Aittala, Janne Hellsten, Jaakko Lehtinen, and Timo Aila. Analyzing and Improving the Image Quality of StyleGAN. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8107–8116, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. 10.1109/CVPR42600.2020.00813. URL https://ieeexplore.ieee.org/document/9156570/.

[23] Florian Lehmann and Daniel Buschek. Autocompletion as a Basic Interaction Concept for User-Centered AI. 2020. 10.18420/MUC2020-WS111-328. URL http://dl.gi.de/handle/20.500.12116/33507. Publisher: Gesellschaft für Informatik e.V.

[24] Liangda Li, Hongbo Deng, Anlei Dong, Yi Chang, Hongyuan Zha, and Ricardo Baeza-Yates. Analyzing User's Sequential Behavior in Query Auto-Completion via Markov Processes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 123–132, New York, NY, USA, August 2015. ACM Press. ISBN 978-1-4503-3621-5. 10.1145/2766462.2767723. URL https://doi.org/10.1145/2766462.2767723.

[25] Yanen Li, Anlei Dong, Hongning Wang, Hongbo Deng, Yi Chang, and ChengXiang Zhai. A two-dimensional click model for query auto-completion. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, SIGIR '14, pages 455–464, New York, NY, USA, July 2014. ACM Press. ISBN 978-1-4503-2257-7. 10.1145/2600428.2609571. URL https://doi.org/10.1145/2600428.2609571.

[26] Antonios Liapis. Can computers foster human users' creativity? theory and praxis of mixed-initiative co-creativity. *Digital Culture & Education*, 8(2):136–153, 2016. URL https://www.um.edu.mt/library/oar/handle/123456789/29476.

[27] Chunbin Lin, Jiaheng Lu, Tok Wang Ling, and Bogdan Cautis. LotusX: A Position-Aware XML Graphical Search System with Auto-Completion. In *2012 IEEE 28th International Conference on Data Engineering*, pages 1265–1268, Washington, DC, USA, April 2012. IEEE. 10.1109/ICDE.2012.123. ISSN: 2375-026X.

[28] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient Estimation of Word Representations in Vector Space. *arXiv:1301.3781 [cs]*, September 2013. URL http://arxiv.org/abs/1301.3781. arXiv:1301.3781.

[29] Bhaskar Mitra and Nick Craswell. Query Auto-Completion for Rare Prefixes. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, CIKM '15, pages 1755–1758, New York, NY, USA, October 2015. ACM Press. ISBN 978-1-4503-3794-6. 10.1145/2806416.2806599. URL https://doi.org/10.1145/2806416.2806599.

[30] Bhaskar Mitra, Milad Shokouhi, Filip Radlinski, and Katja Hofmann. On user interactions with query auto-completion. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, SIGIR '14, pages 1055–1058, New York, NY, USA, July 2014. ACM Press. ISBN 978-1-4503-2257-7. 10.1145/2600428.2609508. URL https://doi.org/10.1145/2600428.2609508.

[31] Kamyar Nazeri, Eric Ng, Tony Joseph, Faisal Qureshi, and Mehran Ebrahimi. EdgeConnect: Structure Guided Image Inpainting using Edge Prediction. In *2019 IEEE/CVF International Conference on Computer Vision*

[32] Vinoth Pandian and Sarah Suleri. BlackBox Toolkit: Intelligent Assistance to UI Design. In *CHI'20, Workshop on Artificial Intelligence for HCI: A Modern Approach*, April 2020.

[33] Dae Hoon Park and Rikio Chiba. A Neural Language Model for Query Auto-Completion. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '17, pages 1189–1192, New York, NY, USA, August 2017. ACM Press. ISBN 978-1-4503-5022-8. 10.1145/3077136.3080758. URL https://doi.org/10.1145/3077136.3080758.

[34] Philip Quinn and Shumin Zhai. A Cost-Benefit Study of Text Entry Suggestion Interaction. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 83–88, New York, NY, USA, May 2016. ACM Press. ISBN 978-1-4503-3362-7. 10.1145/2858036.2858305. URL https://dl.acm.org/doi/10.1145/2858036.2858305.

[35] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. page 24, 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

[36] Milad Shokouhi. Learning to personalize query auto-completion. In *Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '13, pages 103–112, New York, NY, USA, July 2013. ACM Press. ISBN 978-1-4503-2034-4. 10.1145/2484028.2484076. URL https://doi.org/10.1145/2484028.2484076.

[37] Milad Shokouhi and Kira Radinsky. Time-sensitive query auto-completion. In *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '12, pages 601–610, New York, NY, USA, August 2012. ACM Press. ISBN 978-1-4503-1472-5. 10.1145/2348283.2348364. URL https://doi.org/10.1145/2348283.2348364.

[38] Alexey Svyatkovskiy, Ying Zhao, Shengyu Fu, and Neel Sundaresan. Pythia: AI-assisted Code Completion System. In *KDD '19: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, pages 2727–2735, New York, NY, USA, July 2019. ACM Press. ISBN 978-1-4503-6201-6. 10.1145/3292500.3330699. URL https://doi.org/10.1145/3292500.3330699.

[39] Caglar Tirkaz, Berrin Yanikoglu, and T. Metin Sezgin. Sketched symbol recognition with auto-completion. *Pattern Recognition*, 45(11):3926–3937, November 2012. ISSN 0031-3203. 10.1016/j.patcog.2012.04.026. URL http://www.sciencedirect.com/science/article/pii/S0031320312002063.

[40] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, pages 6000–6010, Long Beach, California, USA, December 2017. Curran Associates Inc. ISBN 978-1-5108-6096-4.

[41] Yi Wang, Xin Tao, Xiaojuan Qi, Xiaoyong Shen, and Jiaya Jia. Image Inpainting via Generative Multi-column Convolutional

Workshop (ICCVW)*, pages 3265–3274, Seoul, Korea (South), October 2019. IEEE. ISBN 978-1-72815-023-9. 10.1109/ICCVW.2019.00408. URL https://ieeexplore.ieee.org/document/9022543/.

Neural Networks. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018*. Curran Associates Inc., 2018. 10.5555/3326943.3326974. URL http://arxiv.org/abs/1810.08771. arXiv: 1810.08771.

[42] Qian Yang, John Zimmerman, Aaron Steinfeld, and Anthony Tomasic. Planning Adaptive Mobile Experiences When Wireframing. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems – DIS '16*, pages 565–576, New York, NY, USA, 2016. ACM Press. ISBN 978-1-4503-4031-1. 10.1145/2901790.2901858. URL http://dl.acm.org/citation.cfm?doid=2901790.2901858.

[43] Qian Yang, Nikola Banovic, and John Zimmerman. Mapping Machine Learning Advances from HCI Research to Reveal Starting Places for Design Innovation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems – CHI '18*, pages 1–11, New York, NY, USA, 2018. ACM Press. ISBN 978-1-4503-5620-6. 10.1145/3173574.3173704. URL http://dl.acm.org/citation.cfm?doid=3173574.3173704.

[44] Qian Yang, Aaron Steinfeld, Carolyn Rosé, and John Zimmerman. Re-examining Whether, Why, and How Human-AI Interaction Is Uniquely Difficult to Design. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, pages 1–13, New York, NY, USA, April 2020. ACM Press. ISBN 978-1-4503-6708-0. 10.1145/3313831.3376301. URL https://doi.org/10.1145/3313831.3376301.

[45] Georgios N Yannakakis, Antonios Liapis, and Constantine Alexopoulos. Mixed-initiative co-creativity. In *9th International Conference on the Foundations of Digital Games*, page 8, 2014. URL https://www.um.edu.mt/library/oar//handle/123456789/29459.

[46] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas S Huang. Generative Image Inpainting with Contextual Attention. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5505–5514, Salt Lake City, UT, USA, 2018. 10.1109/CVPR.2018.00577. URL https://ieeexplore.ieee.org/document/8578675.

[47] Aston Zhang, Amit Goyal, Weize Kong, Hongbo Deng, Anlei Dong, Yi Chang, Carl A. Gunter, and Jiawei Han. adaQAC: Adaptive Query Auto-Completion via Implicit Negative Feedback. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '15, pages 143–152, New York, NY, USA, August 2015. ACM Press. ISBN 978-1-4503-3621-5. 10.1145/2766462.2767697. URL https://doi.org/10.1145/2766462.2767697.

[48] Aston Zhang, Amit Goyal, Ricardo Baeza-Yates, Yi Chang, Jiawei Han, Carl A. Gunter, and Hongbo Deng. Towards Mobile Query Auto-Completion: An Efficient Mobile Application-Aware Approach. In *Proceedings of the 25th International Conference on World Wide Web*, WWW '16, pages 579–590, Montréal, Québec, Canada, April 2016. International World Wide Web Conferences Steering Committee. ISBN 978-1-4503-4143-1. 10.1145/2872427.2882977. URL https://doi.org/10.1145/2872427.2882977.

[49] Shuo Zhang and Krisztian Balog. Auto-completion for Data Cells in Relational Tables. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, pages 761–770, New York, NY,

USA, November 2019. ACM Press. ISBN 978-1-4503-6976-3. 10.1145/3357384.3357932. URL https://doi.org/10.1145/3357384.3357932.

[50] Xinyi Zhang and Michiel van de Panne. Data-driven autocompletion for keyframe animation. In *Proceedings of the 11th Annual International Conference on Motion, Interaction, and Games*, MIG '18, pages 1–11, New York, NY, USA, November 2018. ACM Press. ISBN 978-1-4503-6015-9. 10.1145/3274247.3274502. URL https://doi.org/10.1145/3274247.3274502.
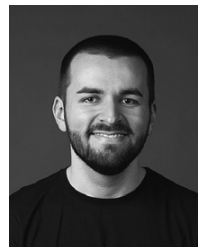
Bionotes

Florian Lehmann
Research Group HCI + AI, Department of Computer Science, University of Bayreuth, Bayreuth, Germany
florian.lehmann@uni-bayreuth.de

Florian Lehmann is a doctoral researcher focusing on research combining Human-Computer Interaction (HCI) and Artificial Intelligence (AI). He is working in a junior research group led by Daniel Buschek at the University of Bayreuth, Germany. He received his master's degree in Human-Computer Interaction from LMU Munich. He has also a background in interactive media and electronics. In his research, he investigates the interaction between humans and intelligent systems such as computational generative systems.
Bibliography e. g. see Google Scholar: https://scholar.google.com/citations?user=akHOQhoAAAAJ&sortby=pubdate

Daniel Buschek
Research Group HCI + AI, Department of Computer Science, University of Bayreuth, Bayreuth, Germany
daniel.buschek@uni-bayreuth.de

Daniel Buschek leads a junior research group at the intersection of Human-Computer Interaction and Machine Learning / Artificial Intelligence at the University of Bayreuth, Germany. Previously, he worked at the Media Informatics group at LMU Munich, where he had also completed his doctoral studies, including research stays at the University of Glasgow and Aalto University, Helsinki. In his research, he combines HCI and AI to create novel user interfaces that enable people to use digital technology in more effective, efficient, expressive, explainable, and secure ways. In short, he is interested in both "AI for better UIs" and "better UIs for AI".
Bibliography e. g. see Google Scholar: https://scholar.google.de/citations?user=TsVkUBwAAAAJ