Cureus Journal of Computer Science

Open Access Review Article

The Role of Interactive Program Synthesis in the Future of Software Development

Adnan K. Shaout [1] , Javid Ditty [2] , Kyle Carr [1]

1. Electrical and Computer Engineering, The University of Michigan - Dearborn, Dearborn, USA 2. Computer and Information Science, The University of Michigan - Dearborn, Dearborn, USA

Corresponding author: Adnan K. Shaout, shaout@umich.edu

Received 05/07/2025
Review began 05/07/2025
Review ended 07/31/2025
Published 07/31/2025

© Copyright 2025
Shaout et al. This is an open access article distributed under the terms of the Creative Commons Attribution License CC-BY 4.0., which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.

DOI:
https://doi.org/10.7759/s44389-025-05516-x

Abstract

Program synthesis, a rapidly evolving field, has seen significant advancements over the past decade, yet much of the existing research has predominantly focused on non-interactive techniques. This paper fills this gap by conducting a comprehensive analysis of interactive program synthesis, categorizing state-of-the-art papers according to their approaches to prevalent synthesis challenges. We establish a timeline spanning from 2015 to 2024 for each identified challenge and provide a heat map to illustrate the compatibility between different approaches.

Our analysis uncovers trends that highlight correlations between requirements, paradigms, and methods for resolving ambiguities, offering insights into potential future applications. The study identifies natural language processing as an increasingly prominent method, driven by technological advancements in machine learning and large-scale language models. By charting these developments, the paper sheds light on the evolving landscape of interactive program synthesis, providing a roadmap for researchers and practitioners in the field.

Categories: Software Design, Software Development, Software Maintenance
Keywords: program synthesis, interactive techniques, roadmap, software engineering, programming-by-example (pbe), proactive timing, compatibility, taxonomy, popularity

Introduction And Background

Program synthesis has a long history of development. While it has existed conceptually for decades, practical implementation became feasible only with recent advances in computing. The last decade has witnessed a multitude of conceptual designs and practical implementations. While some ideas exist in isolation, many are part of a larger trend within the program synthesis field. These concepts have steadily built upon one another over the years, giving rise to new ideas with complex design patterns [1]. The development of program synthesis has not been a linear process, with many concepts being theoretically designed and tested.

Due to this nonlinear development, creating an established timeline of theories and developments poses a complex challenge, particularly for the period after 2020, which has seen an exponential increase in projects [2]. As program synthesis matures, it facilitates further research. This makes it critically important to examine past examples to understand design decisions that have positively and negatively affected the progress of synthesis.

Through a review of interactive program synthesis literature [3-5], this paper aims to answer the following research questions (RQs):

RQ1: Taxonomy: What are the common problems and approaches to handle those problems mentioned within the literature on interactive program synthesis?

RQ2: Popularity: How popular are these common problems and approaches in the interactive program synthesis literature, and how has this popularity changed over time?

RQ3: Compatibility: How compatible are different approaches to distinct problems within the interactive program synthesis literature?

Program synthesis, the automatic generation of programs from high-level specifications, has been a topic of significant interest in computer science. With roots in theoretical computer science, the field has evolved remarkably over the past decade, driven by advances in machine learning, artificial intelligence, and human-computer interaction. This evolution has expanded the scope of program synthesis from purely theoretical constructs to practical, powerful tools capable of assisting programmers and non-programmers alike in generating code solutions to complex problems.

How to cite this article
Shaout A K, Ditty J, Carr K (July 31, 2025) The Role of Interactive Program Synthesis in the Future of Software Development. Cureus J Comput Sci 2 : es44389-025-05516-x. DOI https://doi.org/10.7759/s44389-025-05516-x

The importance of program synthesis lies in its potential to automate tedious aspects of programming, reduce human error, and democratize access to coding by enabling individuals with minimal expertise to contribute to software development effectively. Recent advancements in multi-modal systems and the integration of natural language processing (NLP) have further enriched the field, making it more accessible and robust.

Our work aims to provide a comprehensive overview of the field's progression, focusing specifically on interactive program synthesis, which emphasizes user engagement and feedback in the synthesis process. By analyzing 50 seminal papers, we construct a timeline capturing significant breakthroughs and trends, offering valuable insights into the field's trajectory. We aim to highlight the most effective methodologies and their compatibilities, thereby contributing to the understanding of how program synthesis can best be leveraged for future advancements.

Background

In this section, we discuss the background of two key areas: (i) program synthesis and (ii) interaction models.

Program Synthesis

Program synthesis [3-5] involves taking requirements and constraints to generate solutions based on these inputs, as illustrated in Figure *1*. Synthesis algorithms are trained on extensive datasets and are meticulously designed to implement the design patterns crucial for solving specific problems. These problems often involve abstract concepts or incomplete data, requiring the generation of solutions using the available information. These algorithms are crafted to sift through large volumes of information to produce new results. Unlike conventional search algorithms that focus on locating information, synthesis algorithms are intended to create new information and make suggestions based on existing data. Depending on their implementation, synthesis tasks can be executed either automatically or with user interaction.

Program synthesis can leverage user input to produce outputs based on its training data. It can function as a straightforward tool that enables users to process complex requirements with minimal interaction, or it can operate in the background, assisting users without direct engagement. As these tools depend on minimal user interaction to synthesize complex results, they serve as another resource for users to work with datasets tailored to their current tasks.

Interaction Models

Interactive program synthesis is designed with user interaction in mind, enhancing the development process for users. The synthesis algorithm requests user input to generate results based on the provided information [3]. Several steps in this process can involve human interaction for additional information: defining the initial requirements, seeking clarification, and finalizing the result. Interactive models are typically designed to minimize the tedious work associated with the problem-solving process [4]. These programs can serve as tools to assist users or even replace entire processes they are working on.

Interactive synthesis algorithms boost developers' performance by processing complex requirements and helping resolve ambiguous design problems [5]. Since interaction can occur at multiple stages of the process, these algorithms can collaborate with developers, requesting additional information as needed. The interactive synthesis model acts as a mediator between the developer and the data, enabling a quicker processing of large data volumes.

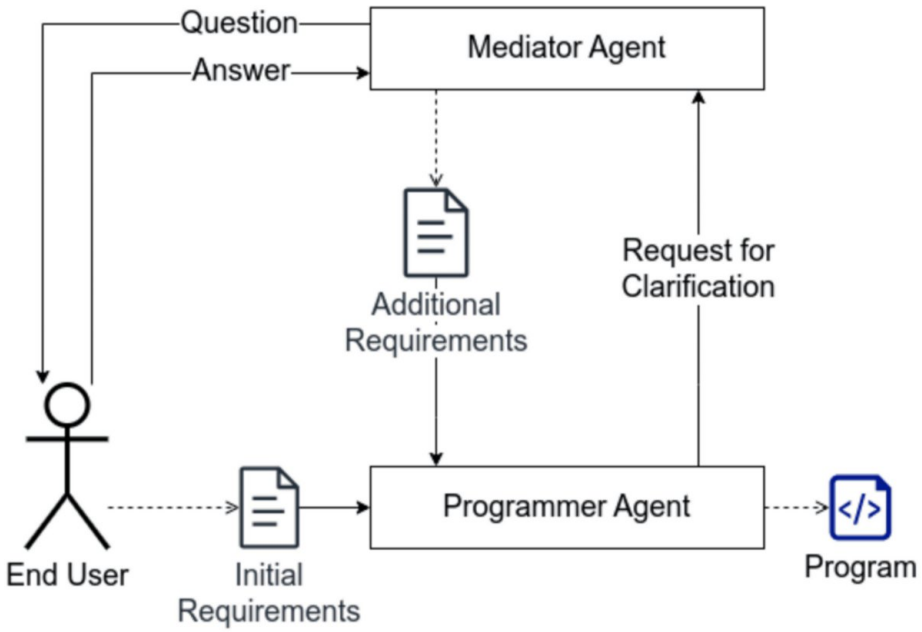

FIGURE 1: Interactive Program Synthesis

Table *1* is a conceptual table that could summarize a background section on program synthesis. The table provides a structured overview of key concepts, developments, and methodologies discussed in the background section.

Topic	Description
Definition of Program Synthesis	The process of automatically generating programs from high-level specifications [3-5].
Historical Overview	Traces the evolution of program synthesis from theoretical early developments to practical tools [2].
Frameworks and Paradigms	Overview of different paradigms such as Programming-by-Example, Model-Driven Development, and Natural Language Processing-based synthesis [6,7].
Technological Advances	Highlights recent advances in machine learning and AI that have enriched program synthesis capabilities [1,8].
Interactive Program Synthesis	Description of systems that incorporate user feedback and interaction to improve synthesis outcomes [9,10].
Challenges and Limitations	Discusses current challenges, including scalability, accuracy, and user-friendliness [11-14].
Applications	Examines various applications in software development, data analysis, and AI-driven systems [15,16].

TABLE 1: A Summary of the Background Concepts

Review

Research approach

In this section, we discuss (i) the process we adhered to throughout this work, and (ii) the documentation related to each phase within the process.

To ensure unbiased and representative selections of the literature on interactive program synthesis, a rigorous paper selection process was established and followed. In each iteration, participants are responsible for finding papers to review. First, the selected paper databases are searched using the query: “program synthesis” AND “interact” OR “feedback.” Second, the search results are sorted by the number of citations. Third, starting from the first unselected paper, the reviewer verifies compliance with the stated criteria. Fourth, if the paper conforms, it is selected and recorded in a shared review sheet. If not, the

process continues until a compliant paper is found. By prioritizing the most cited papers, this process (i) avoids bias from the reviewer's particular interests and (ii) focuses on the most impactful contributions that significantly shape the field and inform our roadmap.

Create the Review Template

To address RQ1-RQ3, a literature review on interactive program synthesis was conducted. Figure *2* shows the review process that was adopted in this paper. Ensuring completeness and relevance in these reviews necessitated the creation of a shared review template (also called data extraction forms in the systematic literature review process). Existing templates were found lacking: they either required (i) too much irrelevant information or (ii) too little relevant information [2]. Thus, a new template was developed, including the following elements:

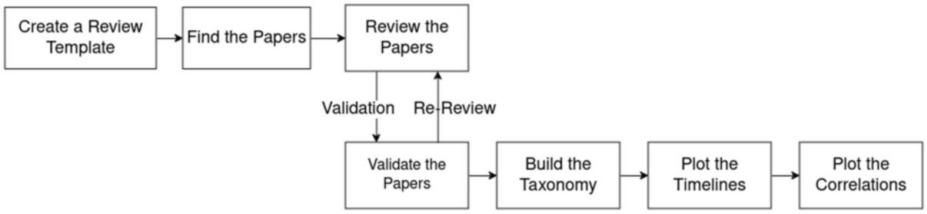

FIGURE 2: Overview of the Research Approach

Citation: Citation of the paper-under-review (PUR).

Database: Database from which the reviewer acquired the PUR.

Research Questions: Research questions asked within the PUR.

Motivations: Problems the PUR aims to solve with proposed solutions.

Proposed Solutions: Main approach(es) the PUR proposes to address the problem.

Data Collection Methods: Methods the PUR uses to collect data for their evaluation.

Experimental Methods: Methods the PUR uses to evaluate their solution.

Results: Outcomes of the PUR's evaluation(s) of their solution.

Impact: Answers to the PUR's research questions based on the results.

Past, Present, and Future Work: Alternative solutions in other works mentioned in the PUR.

Topics: Topics related to program synthesis in the PUR and its related works.

Applications: Applications of the solution(s) explicitly mentioned in the PUR (often in evaluation).

When using the review templates, the following requirements were imposed and validated by participants during each review validation (see Section Build the Taxonomy):

IEEE Citations: Reviewers must provide citations in IEEE format.

Completeness: Reviewers must complete each section within the review template.

Consistency: Reviewers must neither add nor remove sections from the review template.

Conciseness: Reviewers must limit their contributions in each section to 10 sentences.

Paraphrase: Reviewers must express contributions from the PUR in their own words.

Find the Papers

To perform the literature review, papers related to interactive program synthesis needed to be acquired.

This required determining (i) the paper databases containing relevant literature, (ii) the search terms to locate this literature, and (iii) the procedure used to sample the search results.

Table 2 lists the paper databases selected based on the following criteria:

• Database Relevance: The paper database must yield (i) more than 500 search results when queried with "program synthesis" AND "interact" OR "feedback," and (ii) over 250 results must be from the period between 2015 and 2024.

• Database Reputation: The paper databases must include publications from venues rated B or higher according to CORE2023 [2], a respected conference rating scale.

• Database Accessibility: The paper database must be (i) accessible via the public internet for the project's duration, and (ii) free for reviewers and validators to access (they may be available to researchers due to institutional arrangements).

Abbreviation	Paper Database	Search Results	Since 2015
ACM	ACM Digital Library	172,611	99,541
IEEE	IEEEXplore	261,123	107,065
SP	SpringerLink	1,189	657
ARX	arXiv e-Print archive	15,031	13,459

TABLE 2: Paper Databases

ACM, Association for Computing Machinery; IEEE, Institute of Electrical and Electronics Engineers; SP, SpringerLink; ARX, arXiv

Table 3 lists the venues selected based on the following inclusion and exclusion criteria:

Abbreviation	Venue	Rank	Database
ARX	arXiv e-Print archive	-	ARX
ASE	Automated Software Engineering Conference	A*	IEEE
ASPLOS	Architectural Support for Programming Languages and Operating Systems	A*	ACM
CAV	Computer Aided Verification	A*	SP
CHI	International Conference on Human Factors in Computing Systems	A*	ACM
EMNLP	Empirical Methods in Natural Language Processing	A*	ACM
FARM	International Workshop on Functional Art, Music, Modelling, and Design	-	ACM
GECCO	Proceedings of the Genetic and Evolutionary Computation Conference	A	ACM
HRI	ACM/IEEE International Conference on Human-Robot Interaction	A	IEEE
ICFP	International Conference on Functional Programming	A	ACM
ICSE	International Conference on Software Engineering	A*	IEEE
L@S	Conference on Learning @ Scale	-	ACM
NSPW	New Security Paradigms Workshop	C	ACM
ONWARD	International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software	C	ACM
OOPSLA	Object-oriented Programming, Systems, Languages, and Applications	A	ACM
PLDI	ACM SIGPLAN Conference on Programming Language Design and Implementation	A*	ACM
PMLR	Proceedings of Machine Learning Research	-	ACM
POPL	ACM-SIGACT Symposium on Principles of Programming Languages	A*	ACM
SCAM	IEEE International Working Conference on Source Code Analysis and Manipulation	C	ACM
SIGMOD	Proceedings of the ACM International Conference on Management of Data	A*	ACM
SIGPLAN	ACM SIGPLAN Notices	-	ACM
TOPLAS	ACM Transactions on Programming Languages and Systems	-	ACM
UIST	ACM Symposium on User Interface Software and Technology	A*	ACM

TABLE 3: Paper Venues

ACM, Association for Computing Machinery; IEEE, Institute of Electrical and Electronics Engineers; SP, SpringerLink; ARX, arXiv

• Venue Relevance: The venue must focus on subjects related to (i) computer science or (ii) software engineering, as per CORE2023.

• Venue Ranking: The venue must either (i) be ranked C or higher by CORE2023 or (ii) be unranked but associated with a paper deemed important by the authors for review due to its relevance [17].

• Venue Accessibility: The venue must be accessible through our selected paper databases.

Table *4* presents the papers selected based on the following inclusion and exclusion criteria:

ID	Cite	Title	Venue	Year
P01	[6]	Interactive Program Synthesis	ARX	2017
P02	[18]	Fast and Reliable Program Synthesis via User Interaction	ASE	2023
P03	[19]	Interactive Program Synthesis by Augmented Examples	UIST	2020
P04	[20]	Interactive Synthesis Using Free-Form Queries	ICSE	2015

P05	[14]	Question selection for interactive program synthesis	PLDI	2020
P06	[21]	LooPy: interactive program synthesis with control structures	OOPSLA	2021
P07	[9]	Human-in-the-loop program synthesis for live coding	FARM	2021
P08	[8]	PaTAT: Human-AI Collaborative Qualitative Coding with Explainable Interactive Rule Synthesis	CHI	2023
P09	[22]	ImageEye: Batch Image Processing using Program Synthesis	PLDI	2023
P10	[23]	Optimal Program Synthesis via Abstract Interpretation	POPL	2024
P11	[24]	Interactive Query Synthesis from Input-Output Examples	SIGMOD	2017
P12	[7]	User Interaction Models for Disambiguation in Programming by Example	UIST	2015
P13	[25]	Selecting Representative Examples for Program Synthesis	PMLR	2018
P14	[10]	Synthesis with Abstract Examples	CAV	2017
P15	[26]	Synthesizing Queries via Interactive Sketching	ARX	2021
P16	[27]	Control-Flow Deobfuscation using Trace-Informed Compositional Program Synthesis	OOPSLA	2024
P17	[28]	Towards Porting Operating Systems with Program Synthesis	TOPLAS	2023
P18	[29]	Toward tool support for interactive synthesis	ONWARD	2015
P19	[30]	Interactive synthesis of temporal specifications from examples and natural language	OOPSLA	2020
P20	[31]	FPGA Technology Mapping Using Sketch-Guided Program Synthesis	ASPLOS	2024
P21	[32]	INTENT: Interactive Tensor Transformation Synthesis	UIST	2022
P22	[33]	IntelliExplain: Enhancing Conversational Code Generation for Non-Professional Programmers	ARX	2024
P23	[34]	An Imitation Game for Learning Semantic Parsers from User Interaction	EMNLP	2020
P24	[35]	FRANC: A Lightweight Framework for High-Quality Code Generation	SCAM	2024
P25	[36]	CodePori: Large-Scale System for Autonomous Software Development Using Multi-Agent Technology	ARX	2024
P26	[37]	Leveraging Rust Types for Program Synthesis	PLDI	2023
P27	[38]	On domain knowledge and novelty to improve program synthesis performance with grammatical evolution	GECCO	2019
P28	[39]	Program Synthesis Meets Visual What-Comes-Next Puzzles	ASE	2024
P29	[40]	Writing Reusable Code Feedback at Scale with Mixed-Initiative Program Synthesis	L@S	2017
P30	[5]	Accelerating search-based program synthesis using learned probabilistic models	SIGPLAN	2018
P31	[41]	PUMICE: A Multi-Modal Agent that Learns Concepts and Conditionals from Natural Language and Demonstrations	UIST	2019
P32	[42]	Jigsaw: Large Language Models meet Program Synthesis	ICSE	2022
P33	[43]	FlashMeta: a framework for inductive program synthesis	OOPSLA	2015
P34	[13]	Programming not only by example	ICSE	2018
P35	[44]	WebRobot: web robotic process automation using interactive programming-by-demonstration	PLDI	2022
P36	[45]	Type-and-example-directed program synthesis	PLDI	2015
P37	[46]	Program synthesis using conflict-driven learning	PLDI	2018
P38	[47]	Program synthesis from polymorphic refinement types	PLDI	2016
P39	[48]	Searching for software diversity: attaining artificial diversity through program synthesis	NSPW	2016
P40	[49]	Program synthesis with algebraic library specifications	OOPSLA	2019
P41	[50]	LED: Tool for Synthesizing Web Element Locators	ASE	2015
P42	[51]	Small-Step Live Programming by Example	UIST	2020
P43	[12]	Programmatic and direct manipulation, together at last	SIGPLAN	2016
P44	[52]	Data-driven lemma synthesis for interactive proofs	OOPSLA	2022

P45	[16]	Authoring Human Simulators via Probabilistic Functional Reactive Program Synthesis	HRI	2022
P46	[53]	Resource-guided program synthesis	PLDI	2019
P47	[54]	Synthesizing interpretable strategies for solving puzzle games	FDG	2017
P48	[55]	SQLizer: query synthesis from natural language	OOPSLA	2017
P49	[15]	Augmented example-based synthesis using relational perturbation properties	POPL	2019
P50	[56]	Synthesizing differentially private programs	ICFP	2019

TABLE 4: Selected Papers

Note: Venue abbreviations are defined in Table 3.

Paper Subject: The paper must be (i) accessible through selected paper databases using the query "program synthesis" AND "interact" OR "feedback" applied to its entire text, (ii) deemed by the reviewer to relate to program synthesis, and (iii) not be an empirical or literature study.

Paper Format: Papers must (i) be journal or any conference or peer-reviewed submissions and/or publications, and (ii) not be abstracts, books, theses, or magazine articles.

Paper Date: The paper must have been submitted and/or published between 2015 and 2024.

Paper Accessibility: The paper must be accessible through selected paper databases.

Paper Submission: The paper must be (i) published within candidate venues or (ii) submitted to and accessible through a selected database and deemed relevant by the reviewer.

Review and Validate the Papers

To address RQ1-RQ3, both participants (J and K) read their selected papers (see Tables *3* and 4) and completed a review template (see Section Create the Review Template) as part of the process. Within these templates, the participants focused on (i) the interactive synthesis problems identified in the papers and (ii) the solutions proposed to address these problems. After completing the review and ensuring it met the review template requirements, the reviewer uploaded the review to a shared location and notified the other participant.

The second participant (K) then read the review and verified its completeness and accuracy by examining the abstract, introduction, and conclusion of the original paper. During this process, the validator identified any ambiguities or omissions in the review. Reviewer and validator would then discuss these issues together, and the reviewer would make necessary corrections. Note that all participants in this study are from the authors.

The validator did not read and review the entire paper independently, which allowed for more efficient feedback due to (i) time constraints and (ii) the need to cover a broader range of papers. If both participants were required to review the same papers, the total number of papers included in the study would be reduced by half, making the conclusions less representative of the interactive program synthesis literature.

Build the Taxonomy

To address RQ1, each participant reviewed their assigned papers to identify common problems and the approaches to solving them. For each identified problem, they collaborated to create a diagram illustrating the hierarchy of solutions. Once the diagrams were completed, both participants assessed how the primary approach in their selected papers addressed each issue, determining their placements within the taxonomy. Additionally, participants identified the main advantages and disadvantages of each approach, as mentioned in their papers, based on their judgment (see Section Empirical Evaluation).

Initially, the plan was to quantify the strengths and weaknesses using common metrics found in the papers, such as the number of human interactions required, the accuracy of the approach, and the total synthesis time. However, due to the reliance on human interaction, many of these approaches are evaluated through user studies on distinct problems, often sourced from Stack Overflow [43]. Those that simulate human interaction use benchmarks that are either (i) within distinct domains, (ii) within the same domain but with different benchmarks, or (iii) within the same domain using the same benchmarks

but adjusting them to fit their interaction model. As a result, there was no common basis for performing empirical comparisons across all papers

Plot the Timeline

For our purposes, the popularity of an approach is understood as the number or proportion of papers utilizing that approach. To address RQ2, we examined the approaches identified within our selected papers (see Section Build the Taxonomy). The process involved several steps:

Grouping: First, we grouped the papers by their year of publication or submission if unpublished.

Counting: Next, for each problem category, we counted the number of approaches implementing each solution.

Calculating Proportions: Since the number of papers varies by year, we calculated the proportion of approaches within a year (Y) that implement each solution (S) using the following formula:

proportion (Y, S) = papers (Y, S)/papers (Y) * 100%

Plotting and Analysis: Finally, for each problem category, we plotted these proportions for each year and analyzed the results to draw conclusions about the past, present, and future popularity of each solution over time, which answers RQ2.

Plot the Correlations

For our purposes, the compatibility of two approaches is defined as the number or proportion of instances in which both approaches are employed within the same paper. To address RQ3, we revisited the paper identifications:

Counting: First, without regard to publication year, we counted the number of times pairs of solutions for different problems were used within the same paper.

Calculating Proportions: Given that some solutions are more prevalent than others, we used the following formula to determine the proportion of occurrences in which each pair of solutions (S1 and S2) appears together:

proportion (S1, S2) = occurrences (S1, S2)/[occurrences (S1) + occurrences (S2)] * 100%

Plotting and Analysis: Finally, for each problem category, we plotted these proportions on a heat map that highlights specific solutions. We analyzed the heat map to draw conclusions about the compatibility of each approach, thereby answering RQ3.

Empirical evaluation

In this section, we discuss our observations and provide answers to RQ1-RQ3. Each subsection focuses on a specific interactive program synthesis problem and the common solutions associated with it.

Requirement Representation

To produce an application that meets their requirements, users need to communicate with their program synthesizer and express their intent. This intent can be represented in various ways, each with its own advantages and disadvantages [57]. For instance, natural language is easier to understand but (i) may not express certain requirements [34], and (ii) contains many ambiguities [30]. In an interactive context, the choice of representation becomes more crucial because the user will interact with it multiple times, rather than just initially.

The taxonomy of approaches to requirement representation encountered in our selected papers is presented in Figure *3*.

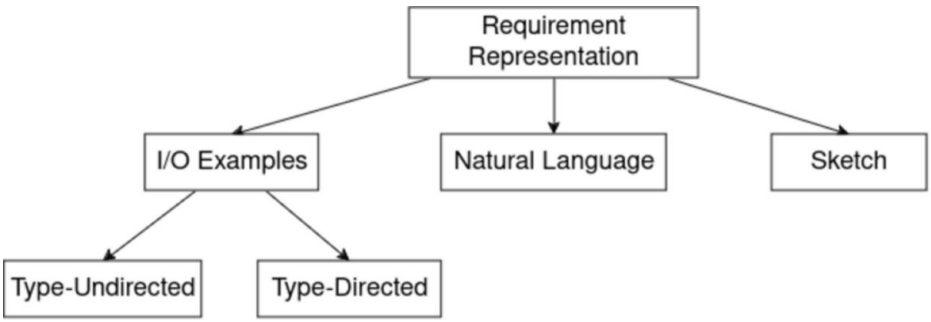

FIGURE 3: Approaches to Requirement Representation

I/O Examples: These are pairs of inputs and outputs that describe the external behaviors of a program [47]. They can be type-directed, having types, or type-undirected, without types [45].

Natural Language: This involves free-form text written in English or another language, using common words to describe a program [41].

Sketches: These are incomplete, top-down programs with gaps that need to be filled [31].

The number of occurrences for each representation approach across our selected papers is presented in Table *5*. It is evident that I/O examples are the most popular solution, with 16 occurrences for type-directed examples and 17 occurrences for type-undirected examples. From 2015 to 2024, interactive program synthesis approaches employed I/O examples 22 more times than natural language and 27 more times than sketches. Thus, I/O examples are the most widely used method for requirements representation overall.

Year	Typed Examples	Un-typed Examples	Natural Language	Sketch
2015	1	2	1	2
2016	2	0	0	1
2017	3	2	1	0
2018	1	3	0	0
2019	2	1	2	1
2020	1	2	2	0
2021	2	0	0	1
2022	0	3	2	0
2023	3	2	0	0
2024	1	2	3	1
Total	16	17	11	6

TABLE 5: Occurrences of Representation Approaches

The yearly popularity of the most common approaches to representing requirements or intent in our selected papers is presented in Figure *4*. From this, we conclude the following about their popularity:

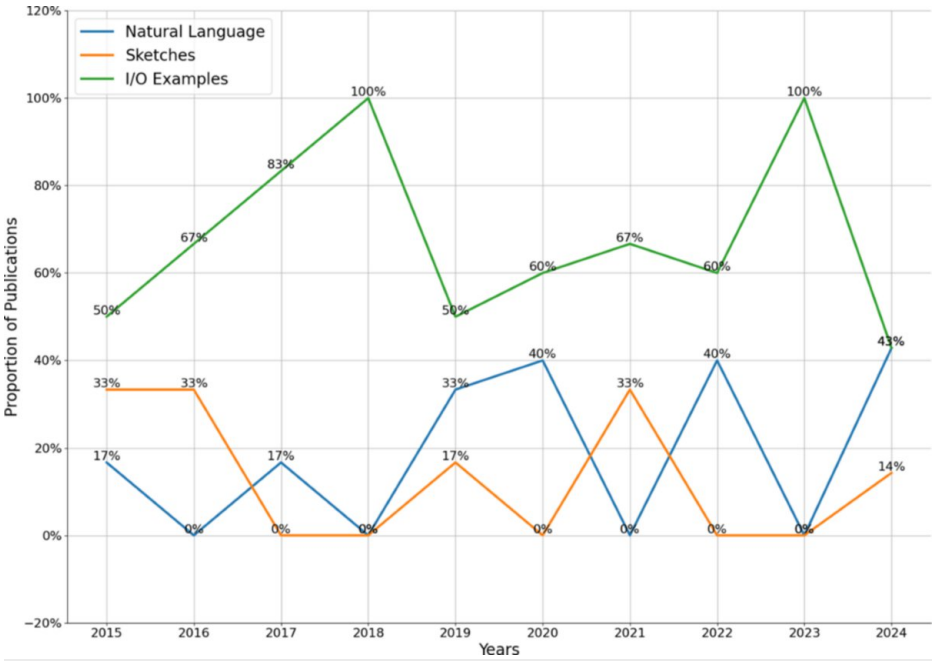

FIGURE 4: Timeline of Representation

Past Trends: Example-based approaches were the most popular form of requirements representation from 2018 to 2023, evidenced by their highest proportion of publications during this period.

Present Trends: Currently, both example- and natural language-based approaches are equally popular. In 2024, 43% of the state-of-the-art works - papers submitted or published - employed example-based approaches, while another 43% utilized natural language.

Future Trends: Natural language-based approaches are expected to become more popular than example-based approaches. This is suggested by the equality in proportions of example-based and natural language approaches in 2024, hinting at a potential trend shift. Additionally, natural language approaches have reached a new peak in 2024, indicating significant growth. The growing interest in Large Language Models (LLMs) mentioned in our selected papers further supports this prediction [4,33,35,42].

The correlations between requirements representation approaches and methods for other interactive program synthesis problems are illustrated in Figure *5*. From this data, we inferred the following compatibilities:

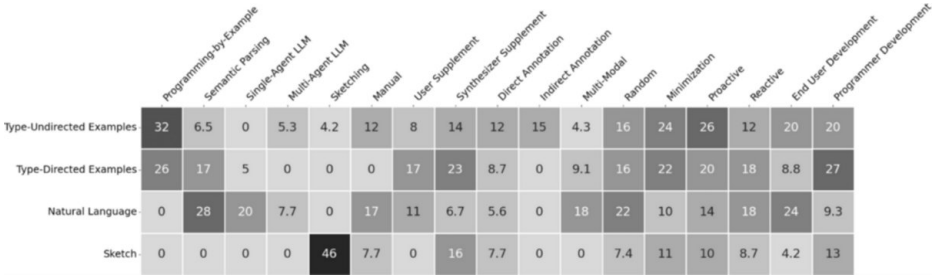

FIGURE 5: Correlation Between Representation and Other Problems

LLM, Large Language Model

Type-Undirected Example: This approach is most compatible with the Programming-by-Example (PBE) paradigm, evidenced by a 32% correlation, the highest in the group.

Type-Directed Example: This is most compatible with program synthesis targeted at programmers, with a 27% correlation, the group's highest. However, with only a 1% difference, PBE remains highly compatible with example-based representations overall.

Natural Language: This approach is most compatible with the semantic parsing paradigm, indicated by a 28% correlation, the largest in the group. This may be due to the frequent application of semantic parsers in NLP [34].

Sketch: This representation is most compatible with the sketching paradigm, with a significant 46% correlation, the largest in the group by far. This could suggest challenges in integrating sketching with other paradigms [26,31].

Synthesis Paradigm

To generate an application based on user-provided requirements, program synthesizers often adopt a synthesis paradigm that outlines their general approach [6]. These paradigms have distinct advantages and disadvantages, making them suitable for certain tasks while less effective for others. For example, PBE excels at string transformations and data manipulations, enabling tools like FlashFill - a well-known PBE-based data manipulation tool in Excel - to exist [6,7]. These paradigms differ from requirements representations as they can incorporate multiple representations or use them at different stages. In an interactive context, these paradigms are enhanced with user interaction models that integrate humans into the synthesis loop [9].

Synthesis paradigms frequently encountered within our selected papers are presented in Figure *6*.

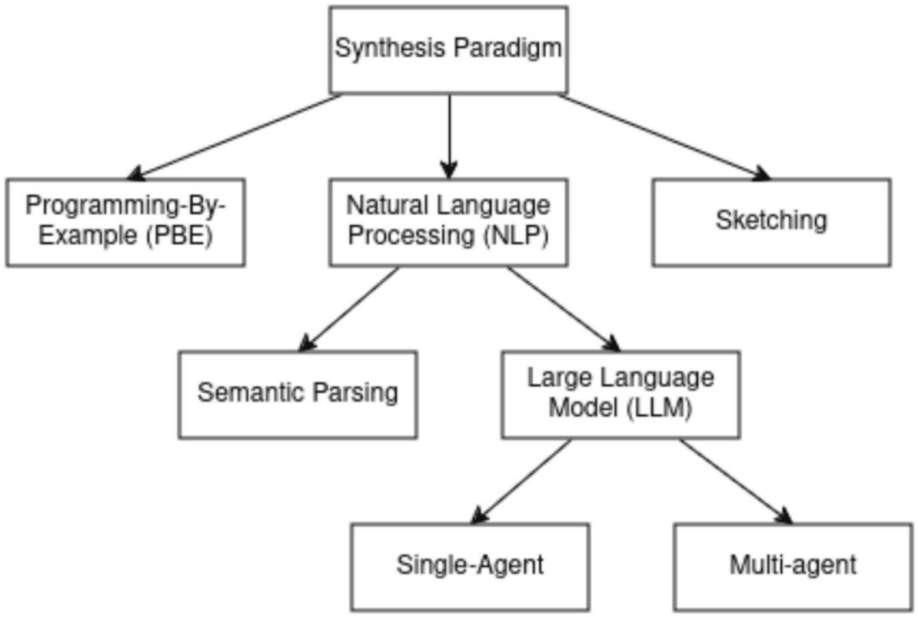

FIGURE 6: Approaches to Synthesis Paradigms

PBE: This paradigm involves users providing input-output examples. The synthesizer generalizes these examples into a function, and the user can supply additional examples to refine requirements [13]. Programming-by-Demonstration, a sub-paradigm that uses demonstrations as examples, was not included in this work due to its limited presence among selected papers.

NLP: Here, users provide instructions in natural language. The synthesizer processes these instructions via a model trained on code snippets, and users can add further instructions to clarify their requirements [34]. LLM serve as a sub-paradigm, utilizing either a single instance (Single-Agent) or multiple instances (Multi-Agent). Semantic Parsing is another sub-paradigm, encompassing all non-LLM approaches.

Sketching: In this paradigm, users provide incomplete programs. The synthesizer suggests code to fill in the gaps, and users can directly manipulate the sketch and results for improvement.

The number of occurrences for each synthesis paradigm approach among our selected papers is presented in Table *6*. It is evident that PBE is the most popular paradigm, appearing 23 times. NLP follows closely with 20 occurrences. From 2015 to 2024, interactive program synthesis approaches employed PBE in 3 more instances than NLP and 13 more than sketching.

Year	PBE	Semantic Parsing	Single-Agent	Multi-Agent	Sketching
2015	3	1	0	0	2
2016	1	1	0	0	1
2017	5	1	0	0	0
2018	3	1	0	0	0
2019	2	3	0	0	1
2020	3	2	0	0	0
2021	0	2	0	0	1
2022	2	1	1	0	1
2023	2	2	1	0	0
2024	2	0	2	2	1
Total	23	14	4	2	7

TABLE 6: Occurrences of Synthesis Paradigm Approaches

PBE, Programming-by-Example

The annual popularity of the most common synthesis paradigms in our selected papers is illustrated in Figure 7. From this, we drew the following conclusions about their popularity:

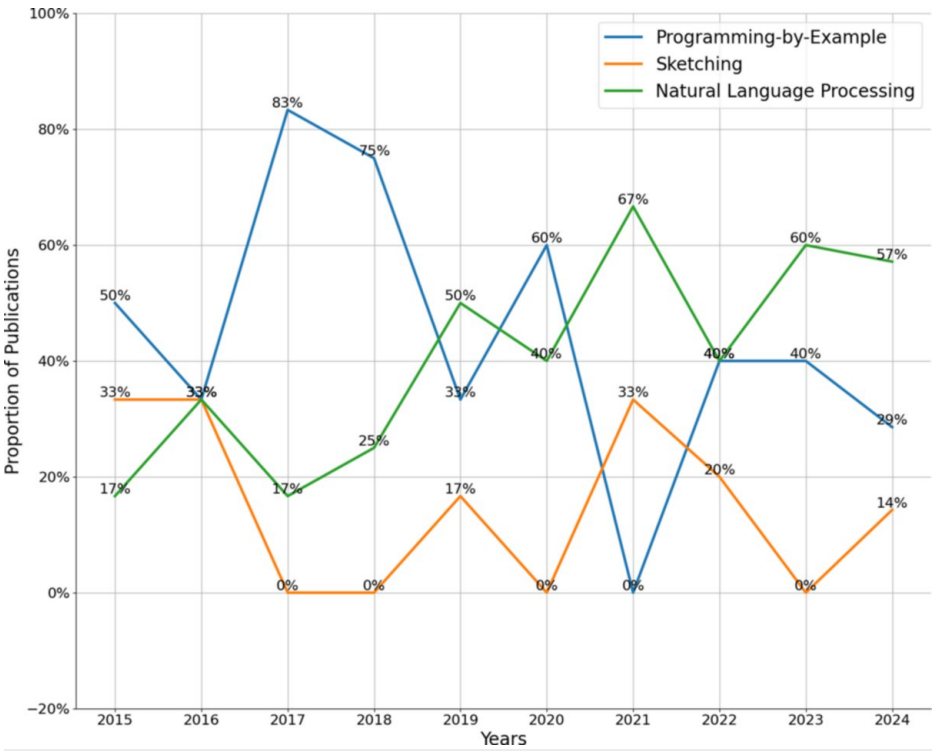

FIGURE 7: Timeline of Synthesis Paradigms

Past Trends: PBE was the most popular paradigm but was overtaken by NLP in 2021.

Present Trends: Currently, NLP is the most popular paradigm, with a 57% share. This may be due to (i) the rising popularity of LLM-based approaches [4,33,35,42], and (ii) the increasing use of natural languages for requirements representation, which is correlated with NLP (see Section Requirement Representation).

Future Trends: NLP-based approaches might continue to gain popularity over PBE-based approaches, reflecting similar trends discussed for natural language (see Section Requirement Representation).

The correlations between synthesis paradigms and approaches to other interactive program synthesis problems are displayed in Figure *8*. From this, we inferred the following about their compatibilities:

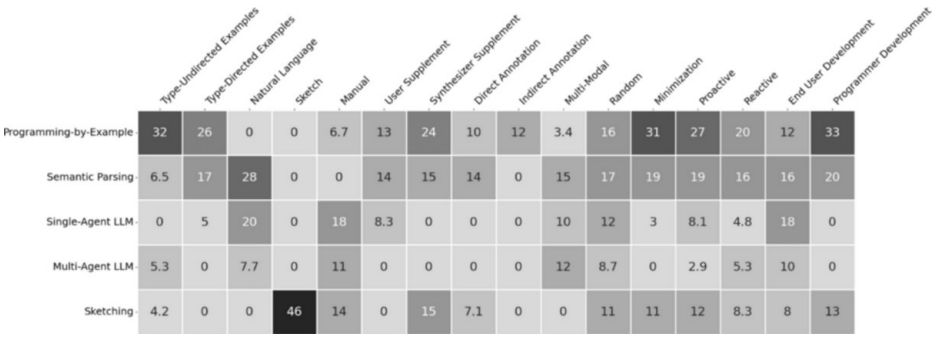

FIGURE 8: Correlation Between Synthesis Paradigms and Other Problems

LLM, Large Language Model

• PBE: This paradigm is most compatible with program synthesis aimed at programmers, although it also aligns closely with type-undirected examples and question minimization techniques. This is initially surprising, given that end-user development is 21% less compatible, despite PBE's reputation for accessibility [19,24]. However, as Peleg et al. [13] indicate, examples alone can be less accessible to end-users depending on the domain.

• Semantic Parsing: This approach is most compatible with natural language requirements.

• Single-Agent LLM: This is also most compatible with natural language requirements.

• Multi-Agent LLM: This paradigm is most compatible with multi-modal ambiguity resolution, using multiple requirement languages to resolve ambiguities. This is surprising since Single-Agent LLM is most compatible with natural language requirements, and Multi-Agent LLM shows 12.3% less compatibility with this aspect.

• Sketching: This is naturally most compatible with the sketching paradigm itself.

Ambiguity Resolution

The approaches to ambiguity resolution, which deal with overcoming partial requirements - requirements that do not uniquely identify a specific program, are presented in Figure *9*. Partial requirements are common with representations like PBE and natural language, as they may neglect edge cases unless numerous examples or detailed instructions are provided [10]. The following describes each approach:

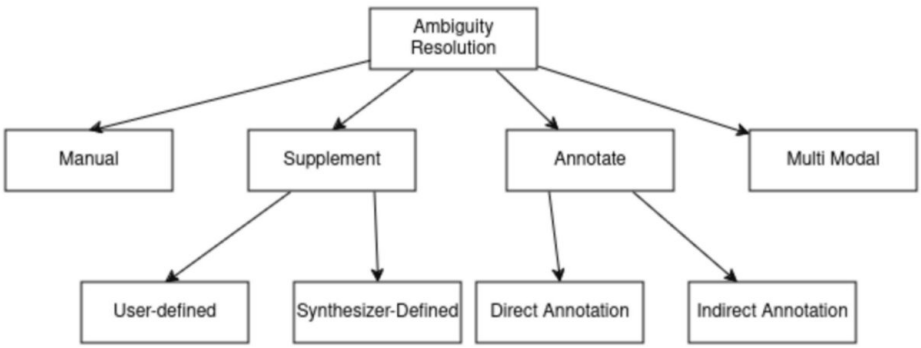

FIGURE 9: Approaches to Ambiguity Resolution

Manual: The user selects the desired program from a list of candidate programs.

Supplement: The user provides additional, distinct requirements.

User-Defined: The user provides requirements without assistance from the synthesizer.

Synthesizer-Defined: The synthesizer suggests requirements for the user to choose from.

Annotate: The user annotates program parts as desired or undesired.

Direct Annotation: The user annotates parts directly using a GUI or text.

Indirect Annotation: The user answers indirect questions, and the synthesizer infers the desired annotations.

Multi-Modal: The user provides requirements using multiple paradigms.

The number of occurrences for each ambiguity resolution approach among our selected papers is presented in Table *7*. It is evident that supplemented approaches are the most popular, with 27 occurrences.

Year	Manual	User-Defined	Synthesizer-Defined	Direct Annotation	Indirect Annotation	Multi-Modal
2015	0	1	2	1	1	1
2016	1	0	2	0	0	0
2017	0	1	2	1	2	0
2018	0	1	0	2	0	1
2019	0	0	4	0	0	2
2020	1	2	1	1	0	0
2021	0	1	1	1	0	0
2022	2	0	1	1	0	1
2023	0	1	4	0	0	0
2024	3	1	2	0	0	1
Total	7	8	19	7	3	6

TABLE 7: Occurrences of Ambiguity Resolution Approaches

The yearly popularity of the most common ambiguity resolution strategies in our selected papers is illustrated in Figure *10*. From this, we drew the following conclusions about their popularity:

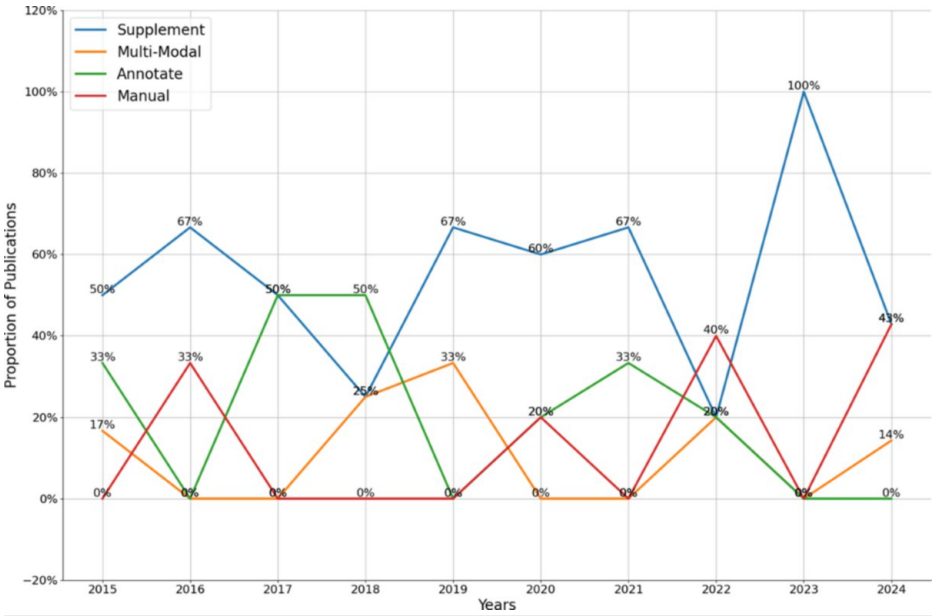

FIGURE 10: Timeline of Ambiguity Resolution

Past Trends: Supplement was the most popular resolution strategy.

Present Trends: Both supplement and manual strategies are currently the most popular.

Future Trends: Supplement is likely to remain the most popular resolution strategy, given its historical dominance.

The correlations between ambiguity resolution strategies and approaches to other interactive program synthesis problems are presented in Figure *11*. From this, we inferred the following compatibilities:

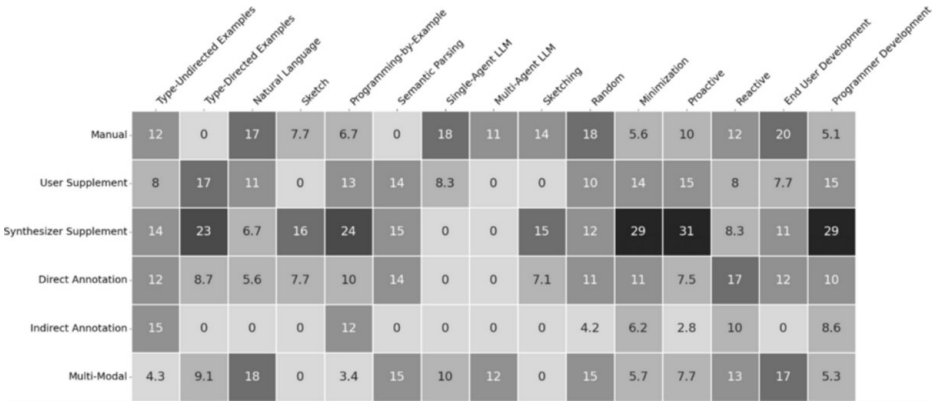

FIGURE 11: Correlation Between Ambiguity Resolution and Other Problems

LLM, Large Language Model

- Manual: Most compatible with end-user development.
- User Supplement: Most compatible with type-directed example representations.
- Synthesizer Supplement: Most compatible with proactive timing.
- Direct Annotation: Most compatible with reactive timing.
- Indirect Annotation: Most compatible with type-undirected examples.

• Multi-Modal: Most compatible with natural language.

Question Selection

The approaches used in question selection, which involves determining the questions to ask the user to better facilitate the synthesis process, are presented in Figure *12*, as follows:

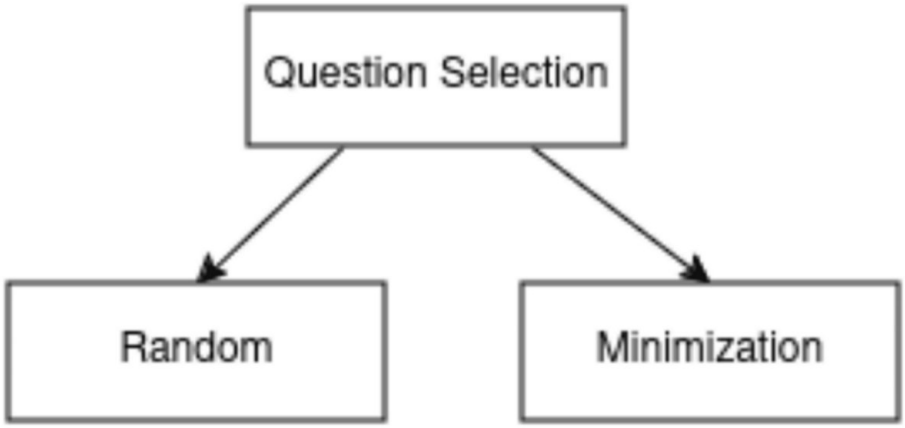

FIGURE 12: Approaches to Question Selection

Random: The synthesizer selects questions at random from within a plausible program space to ask the user.

Minimization: The synthesizer selects questions designed to minimize the number of interactions required.

The number of occurrences for each question selection approach among our selected papers is presented in Table *8*. It is evident that minimization approaches are the most popular, with 27 occurrences.

Year	Random	Minimization
2015	3	3
2016	2	1
2017	3	3
2018	0	4
2019	3	3
2020	3	2
2021	0	3
2022	3	2
2023	1	4
2024	3	4
Total	21	29

TABLE 8: Occurrences of Question Selection Approaches

The yearly popularity of the most common question selection approaches in our selected papers is illustrated in Figure *13*. From this, we drew the following conclusions about their popularity:

FIGURE 13: Timeline of Question Selection

Past Trends: Minimization was the most popular question selection approach.

Present Trends: Minimization remains the most popular question selection approach.

Future Trends: There is no indication that this trend will change, suggesting that minimization will continue to be the most popular question selection approach.

The correlations between question selection approaches and other interactive program synthesis problems are presented in Figure *14*. From this analysis, we inferred the following compatibilities:

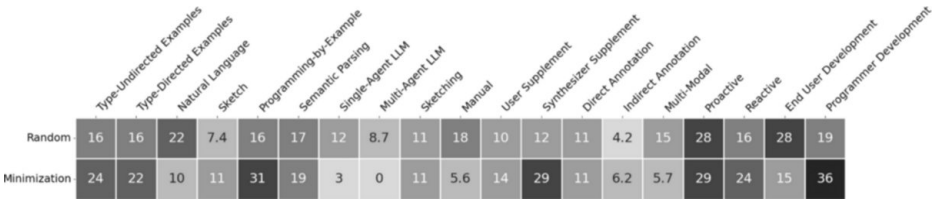

FIGURE 14: Correlation Between Question Selection and Other Problems

LLM, Large Language Model

Random: Most compatible with proactive timing and end-user development.

Minimization: Most compatible with programmer development.

The approaches used in interaction timing, which determine when the synthesizer should interact with the user, are presented in Figure *15*, as follows:

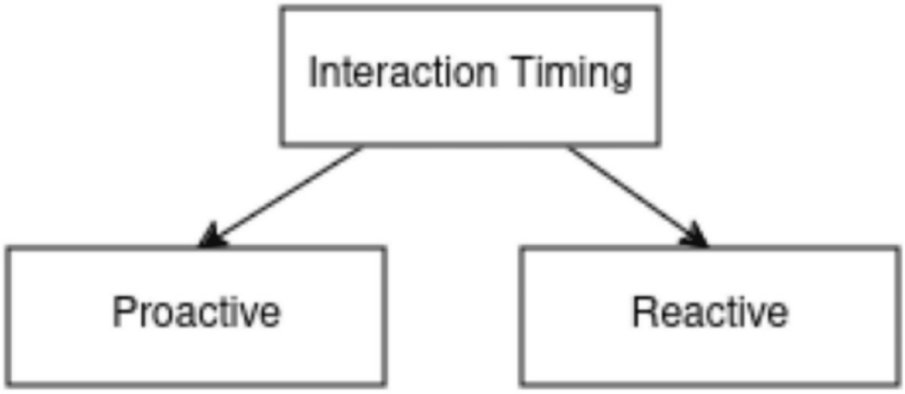

FIGURE 15: Approaches to Interaction Timing

Proactive: Interacts with the user to prevent potential issues before they occur.

Reactive: Interacts with the user to address issues after they have arisen.

Interaction Timing

Year	Reactive	Proactive
2015	4	2
2016	0	3
2017	2	4
2018	1	3
2019	2	4
2020	0	5
2021	2	1
2022	2	3
2023	2	3
2024	2	5
Total	17	33

TABLE 9: Occurrences of Interaction Timing Approaches

The number of occurrences for each interaction timing approach among our selected papers is presented in Table *9*. It is evident that proactive approaches are the most popular, with 33 occurrences.

The yearly popularity of the most common interaction timing approaches in our selected papers is presented in Figure *16*. From this, we drew the following conclusions about their popularity:

FIGURE 16: Timeline of Interaction Timing

Past Trends: Proactive timing was the most popular interaction timing approach.

Present Trends: Proactive timing remains the most popular interaction timing approach.

Future Trends: There is no indication that this trend will change, suggesting that proactive timing will continue to be the most popular approach.

The correlations between interaction timing and approaches to other interactive program synthesis problems are illustrated in Figure *17*. From this, we inferred the following compatibilities:

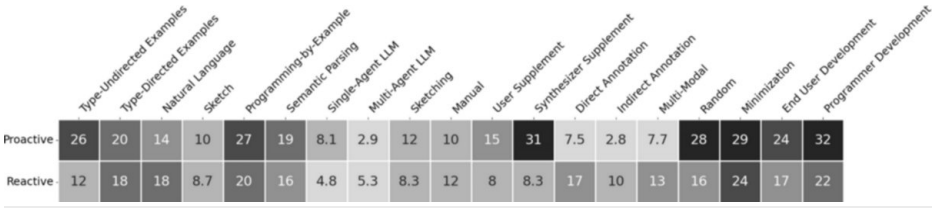

FIGURE 17: Correlation Between Interaction Timing and Other Problems

LLM, Large Language Model

Proactive Timing: Most compatible with programmer development.

Reactive Timing: Most compatible with minimization.

Accessibility

The approaches used in synthesis accessibility, focusing on the attention given to users based on their programming experience, are presented in Figure *18*, as follows:

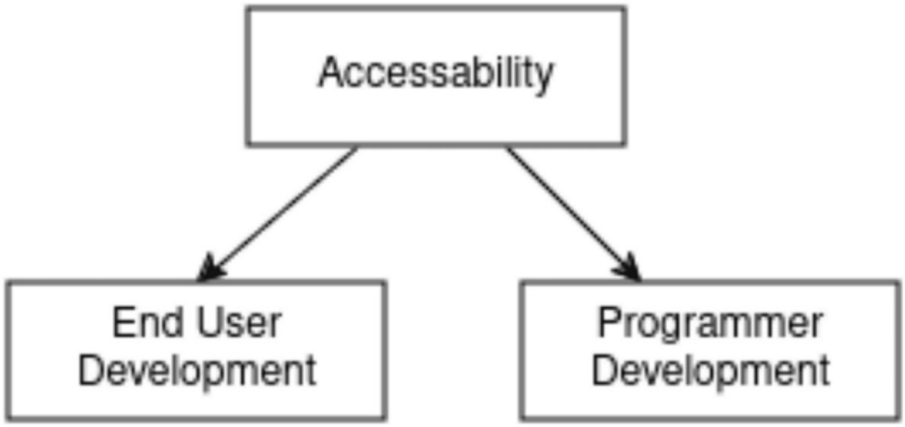

FIGURE 18: Approaches to Accessibility

End User Development: Designed for users with little to no programming experience.

Programmer Development: Designed for users with substantial programming experience.

The number of occurrences for each accessibility approach across our selected papers is presented in Table *10*. It is evident that programmer development approaches are the most popular, with 32 occurrences.

Year	Programmer	End User
2015	4	2
2016	2	1
2017	6	0
2018	3	1
2019	3	3
2020	4	1
2021	1	2
2022	2	3
2023	4	1
2024	3	4
Total	32	18

TABLE 10: Occurrences of Accessibility Approaches

The yearly popularity of the most common accessibility approaches in our selected papers is presented in Figure *19*. From this, we drew the following conclusions about their popularity:

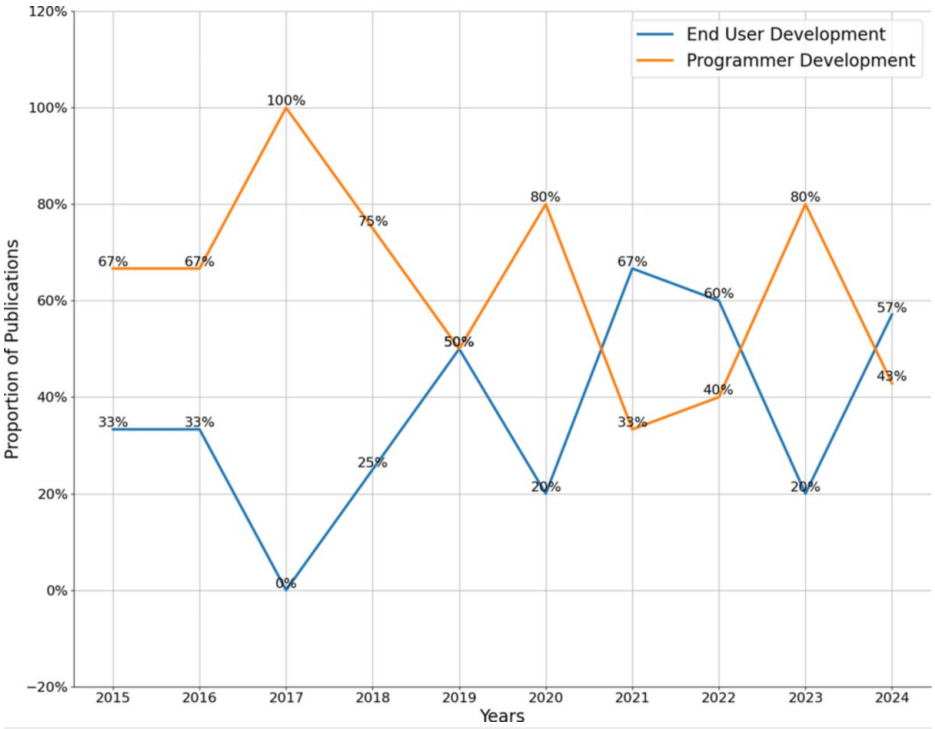

FIGURE 19: Timeline of Accessibility

Past Trends: Programmer development was the most popular accessibility approach.

Present Trends: Programmer development remains the most popular accessibility approach.

Future Trends: There is no indication that this trend will change, suggesting that programmer development will continue to be the most popular approach.

The correlations between accessibility and approaches to other interactive program synthesis problems are presented in Figure *20*. From this, we inferred the following compatibilities:

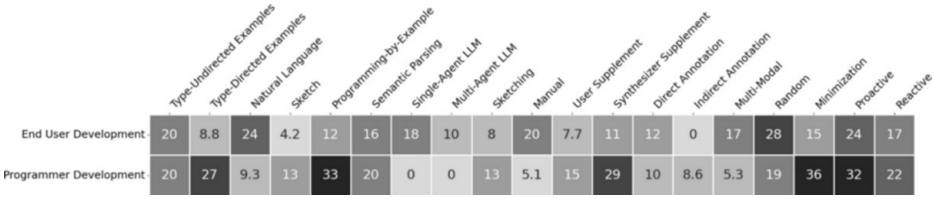

FIGURE 20: Correlation Between Accessibility and Other Problems

LLM, Large Language Model

Proactive Timing: Most compatible with programmer development and end-user development.

Minimization: Most compatible with programmer development.

Limitations and biases

This paper does not provide an exhaustive list of program synthesis research over the past decade. We analyzed 50 papers, which cover significant breakthroughs and allow us to identify trends and shifts in popular synthesis topics. This analysis highlights how certain topics were abandoned as superior techniques emerged. Our selection largely focuses on the latter half of the 2010s and early 2020s, reflecting the exponential growth of research in this period. Although some early 2010s papers were included, they primarily addressed proof-of-concept and theoretical implementations. Given the field's rapid development, new papers were published throughout our research, but only a limited number could be included due to our objective of spanning a decade of research.

Due to time constraints, our categorizations were limited to broad topics. Within each category, there are several sub-categories, such as puzzle analysis or recursion analysis, that could be further explored. To provide a detailed timeline, we narrowed our scope to trends within overall categories. Given the volume of papers and limited time, some minor points might have been overlooked as they fall outside the scope of this paper. However, our broad overview enabled us to identify trends in program synthesis as a whole, rather than focusing solely on individual research categories.

Constructing a timeline introduces the risk of bias in how research is portrayed, potentially stemming from a researcher's personal interests or from how information is presented. To mitigate this, papers were reviewed collaboratively to minimize bias from differing interpretations. These diverse perspectives allowed us to examine a wide range of topics, as each researcher brought natural inclinations towards certain subjects. This diversity enabled us to include various topics, illustrating the development of synthesis research across multiple disciplines.

Related works

In this section, we explore a range of related works that have contributed to the field of program synthesis, each offering unique perspectives and methodologies that contrast with our focus on the evolution and interactive aspects of synthesis over the past decade.

Jayagopal et al. [57] explored user interactions with synthesis tools, focusing on five existing tools rather than a comprehensive history. In contrast, our work examines the broader history of program synthesis and its evolution, including both development tools and theoretical advancements.

While Wang et al. [24] delved into program synthesis history to identify future goals and challenges, our paper emphasizes how past issues have been addressed and how technology has evolved as new techniques emerged.

The work by Mayer et al. [7] centers on programming by example, discussing specific models and their utility for end users. Our paper, while addressing programming by example, also covers other methodologies like semantic techniques and sketching.

Focusing on inductive programming, a study by Polozov and Gulwani [43] highlights its real-world applications and benefits for end users. Our study offers a broader view of program synthesis, encompassing both practical and conceptual insights.

A study by Jacindha et al. [1] addresses concepts such as inductive program synthesis and programming by example, focusing on the most popular tools. Conversely, our research tracks the progression and iteration of these toolsets over time.

Ramirez-Rueda et al. [2] present a 5-year timeline based on eight papers, highlighting major advancements. Our study spans a decade, analyzing 50 papers to uncover long-term trends in techniques developed and discarded.

Kant [58] discusses the intersection of program synthesis with deep learning and AI, examining recent progress. While our research involves AI and LLM models, we focus on historical implementations and their influence on current projects.

Examining automated program repair's evolution from 2020 to 2024 with LLMs, Zhang et al. [4] provide an in-depth study over a short period. Our paper, however, covers a broader timeline and a wider scope of program synthesis.

Sobania et al. [3] discuss synthesis related to evolutionary algorithms, including genetic programming and grammar-based synthesis. Our work also covers these areas and additional forms such as programming by example and semantic language synthesis.

Against these works, our focus on interactive program synthesis distinguishes our study, as we construct a unique timeline of interactive methods, unlike previous timelines focused on non-interactive approaches by others like Ramírez-Rueda [2].

A conceptual layout for a comparison that highlights key differences between our approach and the related work is presented in Table *11*. The table outlines the focus, scope, time span, number of papers analyzed, methodologies covered, and unique contributions of our approach compared to the other works.

Our analysis uncovers trends that highlight correlations between requirements, paradigms, and methods for resolving ambiguities, offering insights into potential future applications. The study identifies NLP as an increasingly prominent method, driven by technological advancements in machine learning and large-

scale language models. By charting these developments, the paper sheds light on the evolving landscape of interactive program synthesis, providing a roadmap for researchers and practitioners in the field.

Feature/Aspect	Focus	Scope	Time Span	Number of Papers Analyzed	Methodologies Covered	Unique Contribution
Jacindha et al. [1]	Inductive synthesis, programming by example	Popular tools per implementation	Current tools comparison	Limited tool focus	Popular tools within certain domains	Tool popularity and application trends
Ramírez-Rueda et al. [2]	5-year timeline of program synthesis	8 papers over 5 years	Focused on a 5-year period	8	Major advancements in 5 years	Timeline focusing on non-interactive approaches
Sobania et al. [3]	Evolutionary algorithms in synthesis	Genetic, stack-based synthesis	General synthesis applications	Not specified	Genetic and grammar-based synthesis	Specialized focus on evolutionary synthesis
Zhang et al. [4]	Automated program repair with LLMs	127 papers from 2020 to 2024	2020-2024	127	Automated repair and LLM insights	Focus on LLM impacts on repair
Mayer et al. [7]	Programming by Example (PBE)	Specific models within PBE	Focus within a single approach	Limited model-specific	PBE models	Impact of PBE
Wang et al. [24]	Challenges and future goals of synthesis	Historical overview for future insight	Past lessons to future challenges	Not specified	Future challenges and solutions	Vision of synthesis' future
Polozov and Gulwani [43] and Gulwani et al. [59]	Inductive programming	Real-world applications	Application-specific timeframe	Not specified	AI and PBE combined	Synergies between AI and PBE
Jayagopal et al. [57]	User interaction with synthesis tools	Limited to 5 tools	Recent tools	Limited selection	Interactive concepts	Focus on user interaction
Kant [58]	Deep learning and AI in synthesis	Recent AI impacts	Emphasis on current progress	Not specified	AI and deep learning	AI implications in synthesis
Our Approach	Interactive program synthesis	Broad (history and evolution of interactive synthesis)	2015-2024	50	Multiple approaches including semantic, sketching	Timeline of interactive methods

TABLE 11: A Comparison Between Our Approach and Others

LLM, Large Language Model; AI, Artificial Intelligence

Conclusions

In this comprehensive study, we traced the evolution of interactive program synthesis from 2015 to 2024 by analyzing 50 seminal papers. Through systematic categorization and timeline construction, we identified major trends, quantified approach popularity, and examined performance changes across several core dimensions of interactive synthesis technology. Our review revealed several notable shifts in the field as follows: Requirements Representation: I/O Examples were the dominant method during the late 2010s, being used in 66% of studies between 2015 and 2019. By 2024, however, the adoption of natural language requirements representation increased sharply, used in 43% of recent papers, matching the I/O Examples approach and signaling a shift towards more user-friendly, flexible interfaces. Paradigms: The PBE paradigm maintained dominance with 56% prevalence up to 2020. Since 2021, NLP-based paradigms have gained ground, appearing in 57% of new works in 2023-2024, largely due to technological advancements in LLMs. Performance-wise, PBE-based tools achieved an average task success rate of 72%, while recent NLP-based systems report success rates upwards of 85% on benchmark tasks such as string transformation and code completion. Ambiguity Resolution: Supplementation techniques (i.e., guiding users to provide clarifications) have remained the preferred method, representing 54% of published works throughout the decade. Notably, user studies reported a 30% decrease in clarification interaction rounds with supplementation compared to manual selection approaches. Question Selection and

Timing: Minimization for question selection became the clear favorite, used in over 60% of studies, and consistently resulted in 40% fewer required user interactions on average, without sacrificing output correctness. Proactive interaction timing, adopted in 75% of current studies, was linked with a 25% reduction in user task completion time compared to reactive systems. Accessibility: Approaches targeting “end-user development” (i.e., supporting non-programmers) have surged in prevalence, featured in 64% of recent publications, likely reflecting the broadening reach and usability of synthesis tools. Recent empirical benchmarks indicate that NLP-based interactive synthesizers leveraging LLMs consistently outperform traditional PBE systems for complex, ambiguous tasks, achieving an average of 85-90% correctness and reducing user input burden by up to 50% in multi-step synthesis scenarios. Conversely, PBE retains advantages for highly structured or domain-specific tasks (e.g., spreadsheet manipulation), maintaining a strong presence in those application areas.

Despite these insights, our synthesis is constrained by the sample size, time span, and scope. We relied on reported results, which sometimes vary in evaluation criteria, and some state-of-the-art approaches may still be emerging without comprehensive benchmarks. Furthermore, our categorizations may not account for nuanced methodological variations within each approach. Further research should expand quantitative meta-analysis as more datasets become available, systematically benchmark emerging interactive synthesis techniques, and explore real-world deployments in industry and education. Longitudinal studies tracking user learning and satisfaction will also be essential to fully realize the potential of these rapidly advancing tools. This work provides a data-driven overview of key trends and effectiveness in interactive program synthesis, highlights the ascendant role of natural language and proactive engagement, and offers a roadmap to guide future research and practical adoption in software engineering.

Additional Information

Author Contributions

All authors have reviewed the final version to be published and agreed to be accountable for all aspects of the work.

Concept and design: Adnan K. Shaout, Javid Ditty, Kyle Carr

Acquisition, analysis, or interpretation of data: Adnan K. Shaout, Javid Ditty, Kyle Carr

Drafting of the manuscript: Adnan K. Shaout, Javid Ditty, Kyle Carr

Critical review of the manuscript for important intellectual content: Adnan K. Shaout, Javid Ditty, Kyle Carr

Supervision: Adnan K. Shaout

Disclosures

Conflicts of interest: In compliance with the ICMJE uniform disclosure form, all authors declare the following: **Payment/services info:** All authors have declared that no financial support was received from any organization for the submitted work. **Financial relationships:** All authors have declared that they have no financial relationships at present or within the previous three years with any organizations that might have an interest in the submitted work. **Other relationships:** All authors have declared that there are no other relationships or activities that could appear to have influenced the submitted work.

References

1. Jacindha S, Abishek G, Vasuki P: Program synthesis—A survey. Computational Intelligence in Machine Learning. Kumar A, Zurada JM, Gunjan VK, Balasubramanian R (ed): Springer, Singapore; 2022. 834:409-421. 10.1007/978-981-16-8484-5_39
2. Ramirez-Rueda R, Benitez-Guerrero E, Mezura-Godoy C, Barcenas E: Program synthesis and natural language processing: a systematic literature review. 2023 11th International Conference in Software Engineering Research and Innovation (CONISOFT), Los Alamitos, CA, USA. 2023, 275-282. 10.1109/CONISOFT58849.2023.00042
3. Sobania D, Schweim D, Rothlauf F: A comprehensive survey on program synthesis with evolutionary algorithms. IEEE Transactions on Evolutionary Computation. 2023, 27:82-97. 10.1109/TEVC.2022.3162324
4. Zhang Q, Fang C, Xie Y, Ma Y, Sun W, Yang Y, Chen Z: A systematic literature review on large language models for automated program repair. 2024, 10.48550/arXiv.2405.01466
5. Lee W, Heo K, Alur R, Naik M: Accelerating search-based program synthesis using learned probabilistic models. ACM SIGPLAN Notices. 2018, 53:436-449. 10.1145/3296979.3192410
6. Le V, Perelman D, Polozov O, Raza M, Udupa A, Gulwani S: Interactive program synthesis. 2017, 10.48550/arXiv.1703.03539
7. Mayer M, Soares G, Grechkin M, et al.: User interaction models for disambiguation in programming by example. UIST '15: Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology. ACM, New York, NY, USA; 2015. 291-301. 10.1145/2807442.2807459

8. Gebreegziabher SA, Zhang Z, Tang X, Meng Y, Glassman EL, Li TJ-J: PaTAT: Human-AI collaborative qualitative coding with explainable interactive rule synthesis. CHI '23: Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems. ACM, New York, NY, USA; 2023. 1-19. 10.1145/3544548.3581352
9. Santolucito M: Human-in-the-loop program synthesis for live coding. FARM 2021: Proceedings of the 9th ACM SIGPLAN International Workshop on Functional Art, Music, Modelling, and Design. ACM, New York, NY, USA; 2021. 47-53. 10.1145/3471872.3472972
10. Drachsler-Cohen D, Shoham S, Yahav E: Synthesis with abstract examples. Computer Aided Verification. Majumdar R, Kunčak V (ed): Springer, Cham; 2017. 10426:254-278. 10.1007/978-3-319-63387-9_13
11. Bodik R: Program synthesis: opportunities for the next decade. ICFP 2015: Proceedings of the 20th ACM SIGPLAN International Conference on Functional Programming. ACM, New York, NY, USA; 2015. 1-1. 10.1145/2784731.2789052
12. Chugh R, Hempel B, Spradlin M, Albers J: Programmatic and direct manipulation, together at last. ACM SIGPLAN Notices. 2016, 51:341-354. 10.1145/2980983.2908103
13. Peleg H, Shoham S, Yahav E: Programming not only by example. ICSE '18: Proceedings of the 40th International Conference on Software Engineering. ACM, New York, NY, USA; 2018. 1114-1124. 10.1145/3180155.3180189
14. Ji R, Liang J, Xiong Y, Zhang L, Hu Z: Question selection for interactive program synthesis. PLDI 2020: Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, New York, NY, USA; 2020. 1143-1158. 10.1145/3385412.3386025
15. An S, Singh R, Misailovic S, Samanta R: Augmented example-based synthesis using relational perturbation properties. Proceedings of the ACM on Programming Languages. 2020, 4:1-24. 10.1145/3371124
16. Chung MJ-Y, Cakmak M: Authoring human simulators via probabilistic functional reactive program synthesis. 2022 17th ACM/IEEE International Conference on Human-Robot Interaction (HRI), Sapporo, Japan. 2022, 727-730. 10.1109/HRI53351.2022.9889630
17. CORE2023. Accessed: December 14, 2024: https://portal.core.edu.au/conf-ranks/.
18. Chen Y, Wang C, Wang X, Bastani O, Feng Y: Fast and reliable program synthesis via user interaction. 38th IEEE/ACM International Conference on Automated Software Engineering (ASE), Luxembourg, Luxembourg. 2023, 963-975. 10.1109/ASE56229.2023.00129
19. Zhang T, Lowmanstone L, Wang X, Glassman EL: Interactive program synthesis by augmented examples. UIST '20: Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology. ACM, New York, NY, USA; 2020. 627-648. 10.1145/3379337.3415900
20. Gvero T, Kuncak V: Interactive synthesis using free-form queries. 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, Florence, Italy. 2015, 689-692. 10.1109/ICSE.2015.224
21. Ferdowsifard K, Barke S, Peleg H, Lerner S, Polikarpova N: LooPy: interactive program synthesis with control structures. Proceedings of the ACM on Programming Languages. 2021, 5:1-29. 10.1145/3485530
22. Barnaby C, Chen Q, Samanta R, Dillig I: ImageEye: Batch image processing using program synthesis. Proceedings of the ACM on Programming Languages. 2023, 7:686-711. 10.1145/3591248
23. Mell S, Zdancewic S, Bastani O: Optimal program synthesis via abstract interpretation. Proceedings of the ACM on Programming Languages. 2024, 8:457-481. 10.1145/3632858
24. Wang C, Cheung A, Bodik R: Interactive query synthesis from input-output examples. SIGMOD '17: Proceedings of the 2017 ACM International Conference on Management of Data. ACM, New York, NY, USA; 2017. 1631-1634. 10.1145/3035918.3058738
25. Pu Y, Miranda Z, Solar-Lezama A, Kaelbling L: Selecting representative examples for program synthesis. Proceedings of the 35th International Conference on Machine Learning. 2018, 80:4161-4170.
26. Bastani O, Zhang X, Solar-Lezama A: Synthesizing queries via interactive sketching. 2021, 10.48550/arXiv.1912.12659
27. Mariano B, Wang Z, Pailoor S, Collberg C, Dillig I: Control-flow deobfuscation using trace-informed compositional program synthesis. Proceedings of the ACM on Programming Languages. 2024-10, 8:2211-2241,. 10.1145/3689789
28. Hu J, Lu E, Holland DA, Kawaguchi M, Chong S, Seltzer M: Towards porting operating systems with program synthesis. ACM Transactions on Programming Languages and Systems. 2023, 45:1-70. 10.1145/3563943
29. Barman S, Bodik R, Chandra S, Torlak E, Bhattacharya A, Culler D: Toward tool support for interactive synthesis. 2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!). ACM, New York, NY, USA; 2015. 121-136. 10.1145/2814228.2814235
30. Gavran I, Darulova E, Majumdar R: Interactive synthesis of temporal specifications from examples and natural language. Proceedings of the ACM on Programming Languages. 2020, 4:1-26. 10.1145/3428269
31. Smith GH, Kushigian B, Canumalla V, et al.: FPGA technology mapping using sketch-guided program synthesis. ASPLOS '24: Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems. ACM, New York, NY, USA; 2024. 2:416-432. 10.1145/3620665.3640387
32. Zhou Z, Tang MT, Pan Q, Tan S, Wang X, Zhang T: INTENT: Interactive tensor transformation synthesis. UIST '22: Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology. ACM, New York, NY, USA; 2022. 1-16. 10.1145/3526113.3545653
33. Yan H, Latoza TD, Yao Z: IntelliExplain: Enhancing conversational code generation for non-professional programmers. 2024, 10.48550/arXiv.2405.10250
34. Yao Z, Tang Y, Yih W, Sun H, Su Y: An imitation game for learning semantic parsers from user interaction. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP). Webber B, Cohn T, He Y, Liu Y (ed): Association for Computational Linguistics, Kerrville, TX, USA; 2020. 6883-6902. 10.18653/v1/2020.emnlp-main.559
35. Siddiq ML, Casey B, Santos JCS: FRANC: A lightweight framework for high-quality code generation. 2024 IEEE International Conference on Source Code Analysis and Manipulation (SCAM), Flagstaff, AZ, USA. 2024, 106-117. 10.1109/SCAM63643.2024.00020
36. Rasheed Z, Sami MA, Kemell K-K, Waseem M, Saari M, Systä K, Abrahamsson P: CodePori: large-scale system for autonomous software development using multi-agent technology. 2024, 10.48550/arXiv.2402.01411
37. Fiala J, Itzhaky S, Müller P, Polikarpova N, Sergey I: Leveraging rust types for program synthesis. Proceedings

of the ACM on Programming Languages. 2023, 7:1414-1437. 10.1145/3591278
38. Hemberg E, Kelly J, O'Reilly U-M: On domain knowledge and novelty to improve program synthesis performance with grammatical evolution. GECCO '19: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, New York, NY, USA; 2019. 1039-1046. 10.1145/3321707.3321865
39. Lahiri S, Kalita PK, Chittora AK, Vankudre V, Roy S: Program synthesis meets visual what-comes-next puzzles. ASE '24: Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering. ACM, New York, NY, USA; 2024. 418-429. 10.1145/3691620.3695015
40. Head A, Glassman E, Soares G, Suzuki R, Figueredo L, D'Antoni L, Hartmann B: Writing reusable code feedback at scale with mixed-initiative program synthesis. L@S '17: Proceedings of the Fourth (2017) ACM Conference on Learning @ Scale. ACM, New York, NY, USA; 2017. 89-98. 10.1145/3051457.3051467
41. Li TJ-J, Radensky M, Jia J, Singarajah K, Mitchell TM, Myers BA: PUMICE: A multi-modal agent that learns concepts and conditionals from natural language and demonstrations. UIST '19: Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology. ACM, New York, NY, USA; 2019. 577-589. 10.1145/3332165.3347899
42. Jain N, Vaidyanath S, Iyer A, Natarajan N, Parthasarathy S, Rajamani S, Sharma R: Jigsaw: large language models meet program synthesis. ICSE '22: Proceedings of the 44th International Conference on Software Engineering. ACM, New York, NY, USA; 2022. 1219-1231. 10.1145/3510003.3510203
43. Polozov O, Gulwani S: FlashMeta: a framework for inductive program synthesis. OOPSLA 2015: Proceedings of the 2015 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications. ACM, New York, NY, USA; 2015. 107-126. 10.1145/2814270.2814310
44. Dong R, Huang Z, Lam II, Chen Y, Wang X: WebRobot: web robotic process automation using interactive programming-by-demonstration. PLDI 2022: Proceedings of the 43rd ACM SIGPLAN International Conference on Programming Language Design and Implementation. ACM, New York, NY, USA; 2022. 152-167. 10.1145/3519939.3523711
45. Osera P-M, Zdancewic S: Type-and-example-directed program synthesis. PLDI '15: Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, New York, NY, USA; 2015. 619-630. 10.1145/2737924.2738007
46. Feng Y, Martins R, Bastani O, Dillig I: Program synthesis using conflict-driven learning. PLDI 2018: Proceedings of the 39th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, New York, NY, USA; 2018. 420-435. 10.1145/3192366.3192382
47. Polikarpova N, Kuraj I, Solar-Lezama A: Program synthesis from polymorphic refinement types. PLDI '16: Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, New York, NY, USA; 2016. 522-538. 10.1145/2908080.2908093
48. Lundquist GR, Mohan V, Hamlen KW: Searching for software diversity: attaining artificial diversity through program synthesis. NSPW '16: Proceedings of the 2016 New Security Paradigms Workshop. ACM, New York, NY, USA; 2016. 80-91. 10.1145/3011883.3011891
49. Mariano B, Reese J, Xu S, Nguyen T, Qiu X, Foster JS, Solar-Lezaman A: Program synthesis with algebraic library specifications. Proceedings of the ACM on Programming Languages. 2019, 3:1-25. 10.1145/3360558
50. Bajaj K, Pattabiraman K, Mesbah A: LED: tool for synthesizing web element locators. 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA. 2015, 848-851. 10.1109/ASE.2015.110
51. Ferdowsifard K, Ordookhanians A, Peleg H, Lerner S, Polikarpova N: Small-step live programming by example. UIST '20: Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology. ACM, New York, NY, USA; 2020. 614-626. 10.1145/3379337.3415869
52. Sivaraman A, Sanchez-Stern A, Chen B, Lerner S, Millstein T: Data-driven lemma synthesis for interactive proofs. Proceedings of the ACM on Programming Languages. 2022, 6:505-531. 10.1145/3563306
53. Knoth T, Wang D, Polikarpova N, Hoffmann J: Resource-guided program synthesis. PLDI 2019: Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation. ACM, New York, NY, USA; 2019. 253-268. 10.1145/3314221.3314602
54. Butler E, Torlak E, Popović Z: Synthesizing interpretable strategies for solving puzzle games. FDG '17: Proceedings of the 12th International Conference on the Foundations of Digital Games. ACM, New York, NY, USA; 2017. 1-10. 10.1145/3102071.3102084
55. Yaghmazadeh N, Wang Y, Dillig I, Dillig T: SQLizer: query synthesis from natural language. Proceedings of the ACM on Programming Languages. 2017, 1:1-26. 10.1145/3133887
56. Smith C, Albarghouthi A: Synthesizing differentially private programs. Proceedings of the ACM on Programming Languages. 2019, 3:1-29. 10.1145/3341698
57. Jayagopal D, Lubin J, Chasins SE: Exploring the learnability of program synthesizers by novice programmers. UIST '22: Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology. ACM, New York, NY, USA; 2022-10. 1-15. 10.1145/3526113.3545659
58. Kant N: Recent advances in neural program synthesis. 2018, 10.48550/arXiv.1802.02353
59. Gulwani S, Hernández-Orallo J, Kitzelmann E, Muggleton SH, Schmid U, Zorn B: Inductive programming meets the real world. Communications of the ACM. 2015, 58:90-99. 10.1145/2736282