Check for updates

Received: 22 November 2021 | Revised: 3 October 2022 | Accepted: 5 November 2022

DOI: 10.1111/jcal.12757

ARTICLE

Journal of Computer Assisted Learning WILEY

Supporting skill integration in an intelligent tutoring system for code tracing

Yun Huang[1,2,3] | Peter Brusilovsky[3] | Julio Guerra[2] | Kenneth Koedinger[1] | Christian Schunn[4]

[1]Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA

[2]Institute of Informatics, Austral University of Chile, Valdivia, Los Ríos, Chile

[3]School of Computing and Information, University of Pittsburgh, Pittsburgh, Pennsylvania, USA

[4]Learning Research and Development Center, University of Pittsburgh, Pittsburgh, Pennsylvania, USA

Correspondence
Yun Huang, Mehuin 1512, Valdivia, Region de los Rios, 5110912, Chile.
Email: yun.huang@uach.cl

Abstract

Background: Skill integration is vital in students' mastery development and is especially prominent in developing code tracing skills which are foundational to programming, an increasingly important area in the current STEM education. However, instructional design to support skill integration in learning technologies has been limited.

Objectives: The current work presents the development and empirical evaluation of instructional design targeting students' difficulties in code tracing particularly in integrating component skills in the Trace Table Tutor (T3), an intelligent tutoring system.

Methods: Beyond the instructional features of active learning, step-level support, and individualized problem selection of intelligent tutoring systems (ITS), the instructional design of T3 (e.g., hints, problem types, problem selection) was optimized to target skill integration based on a domain model where integrative skills were represented as combinations of component skills. We conducted an experimental study in a university-level introductory Python programming course and obtained three findings.

Results and Conclusions: First, the instructional features of the ITS technology support effective learning of code tracing, as evidenced by significant learning gains (medium-to-large effect sizes). Second, performance data supports the existence of integrative skills beyond component skills. Third, an instructional design focused on integrative skills yields learning benefits beyond a design without such focus, such as improving performance efficiency (medium-to-large effect sizes).

Major Takeaways: Our work demonstrates the value of designing for skill integration in learning technologies and the effectiveness of the ITS technology for computing education, as well as provides general implications for designing learning technologies to foster robust learning.

The research in this article was conducted as Yun Huang's doctoral dissertation work at University of Pittsburgh. Her current affiliations are Carnegie Mellon University (primary, postdoctoral fellow) and Austral University of Chile (secondary, visiting researcher), where she conducted further analysis and writing for this research.

This is an open access article under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs License, which permits use and distribution in any medium, provided the original work is properly cited, the use is non-commercial and no modifications or adaptations are made.
© 2022 The Authors. *Journal of Computer Assisted Learning* published by John Wiley & Sons Ltd.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

KEYWORDS
adaptive educational systems, computer science education, domain modelling, instructional design, intelligent tutoring systems, programming education

1 | INTRODUCTION

Skill integration is important and challenging for students' mastery development. Based on research evidence from the science of learning and the science of instruction, Ambrose et al. (2010) proposed a general framework of mastery development: students must acquire component skills, practice integrating skills, and know when to apply skills. In their view, the second phase, skill integration, is a difficult phase with high cognitive load demands where students need to combine component skills with fluency and autonomy. The Knowledge-Learning-Instruction (KLI) framework that aims at bridging instructional decision making and the science of learning (Koedinger, Corbett, & Perfetti, 2012), also stated that integrative knowledge has significance for learning and instruction. In particular, they defined an 'integrative knowledge component' as a knowledge component (KC) that 'integrates or must be integrated (or connected) with other KCs to produce behavior' (p. 771). The notion of *integrative skills*[1] in the current work is based upon this framework. Rigorous empirical evidence of integrative skills can be found in the investigations into the *composition effect*, the phenomenon of the whole being more difficult than the sum of the parts, in mathematics learning (Alibali et al., 2014; Heffernan & Koedinger, 1997). In particular, Heffernan and Koedinger et al. (1997) found that students were significantly worse at translating two-operator algebra story problems into expressions (e.g., 800-40*x*) than they were at translating two closely matched one-operator problems (e.g., with answers 800-*y* and 40*x* separately). They suggested that the students missed a specific integrative skill, namely a recursive grammar rule that indicates expressions (e.g., 40*x*) can be embedded in other expressions (e.g., 800-40*x*).

Despite the importance and difficulty of skill integration, instructional design for skill integration in learning technologies has been limited. Most of the time, the emphasis has been on decomposing complex tasks and domain knowledge into component skills without a deliberate differentiation between component skills used in simple tasks and integrative skills that may be needed. This approach has been taken in the design of many *intelligent tutoring systems* (ITS), a well-established technology proven to be highly effective for skill mastery in various domains (Ritter et al., 2007; VanLehn et al., 2005). An ITS provides step-by-step problem-solving support and individualized deliberate practice on each skill (Koedinger & Anderson, 1993; Lovett, 2001). In particular, *deliberate practice* has been generally established as crucial for expertise acquisition where training is focused on specifically designed tasks with feedback and repetition for improving critical aspects of performance (Ericsson, 2006; Schnackenberg et al., 1998); ITSs can deliver deliberate practice in an optimized form by providing as-needed repetition for each student. Admittedly, mastering each component skill is the first phase towards mastery of the task domain and targeted practice on weak component skills can improve students' overall performance. But to foster more robust learning on more complex and realistic tasks students also need to master skill integration. For example, prior work has shown that some new misconceptions were only revealed when component skills are combined in ways subtly different from how they were typically combined in examples in lectures (Vainio & Sajaniemi, 2007). A few studies have started the investigation into instructional design for skill integration in mathematics learning (Koedinger et al., 2013; Koedinger & McLaughlin, 2010). For example, to better teach students to write algebraic expressions in story problems (i.e., symbolization), Koedinger and McLaughlin (2010) designed substitution problems (e.g., 'Substitute 62-f for b in 62+b.') for learning the integrative skill identified in a prior study (Heffernan & Koedinger, 1997). These substitution problems provide *focused practice* where certain skill requirements are removed from the problem formulation (e.g., comprehending stories) or steps are removed from the interface, so that students could focus *attention* on a single skill or step, following the *Focused Practice Task Design* method (Huang et al., 2021). Students under the substitution condition significantly improved their performance on two-operator story symbolization compared to the control group given the more similar, one-operator story problems instead of substitution problems.

To investigate instructional design for skill integration, we consider an increasingly important domain, programming, where skill integration is critical throughout even introductory courses. In particular, we focus on *code tracing*, a foundational skill in programming that involves stepping through a program to predict values of variables and outputs. Novices struggle with code tracing across variations of how programming constructs are used together (Lister et al., 2004; Sorva, 2012; Stephens-Martinez et al., 2017). For example, executing an addition assignment statement inside a *for* loop for computing a cumulative sum (i.e., integrating an addition assignment statement and a *for* statement) already posed a difficulty for novices and revealed misconceptions that students harboured after standard lectures and exercises according to our prior work (Huang, 2018). Meanwhile, there is some initial evidence of the effectiveness of explicit instructions on skill integration in programming: Muller et al. (2007) and de Raadt (2008) showed that novices who studied using a revised course with explicit emphasis on programming patterns, which typically involve skill integration as we refer to here, exhibited better problem-solving competence than those who studied in a traditional manner.

Meanwhile, the ITS technology is still underused or only used in a partial form in programming education, and one reason may be the

[1]In this work, we use the terms *skill* and *integrative skill* as synonyms for 'knowledge component (KC)' and 'integrative KC'. A KC in the KLI framework (Koedinger, Corbett, & Perfetti, 2012; Koedinger, McLaughlin, & Stamper, 2012) is defined as an acquired unit of cognitive function or structure that can be inferred from performance on a set of related tasks, such as a concept or a skill.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

high demand involved in the construction of an accurate *domain model*, the key component of an ITS. A domain model specifies the skills that a novice must learn to reach expertise and often includes a mapping from a problem or a problem step to the skills. It is the base for a *learner model*, another component of an ITS that maintains dynamic estimations of each student's skill levels for achieving individualized problem selection. A domain model can also inform the design of hints and problem types. Past research showed that instructional (re)design driven by a refined domain model led to significant improvements in learning (Koedinger et al., 2013; Liu & Koedinger, 2017). To realize effective instructional design to support skill integration, an important prior step is to identify the integrative skills and represent them in the domain model. Constructing a domain model for any kind of ITS is challenging, but it is especially challenging for such a complex domain as programming due to its integrative nature. In our prior work (Huang, 2018), we analysed students' errors in Python code tracing and identified a set of important integrative skills apart from component skills. The current work builds on this prior work and evaluates an instructional design for skill integration driven by an *integration-level domain model* that represents the identified integrative skills as combinations of component skills with structural constraints as in Huang (2018).

The paper reports on the development and empirical evaluation of instructional design targeting students' difficulties in Python code tracing particularly in integrating component skills through an ITS, *Trace Table Tutor* (*T3*). We investigated the following research questions (RQs):

> **RQ1.** *Do the instructional features of an ITS support effective learning in code tracing?*
>
> **RQ2.** *Does performance data support the existence of integrative skills?*
>
> **RQ3.** *Does tutored practice focused on skill integration enhance learning?*

RQ1 investigates whether the instructional features of a full-scale ITS, particularly active learning, step-level interaction and feedback, and individualized problem selection, which are the core features of T3, support effective learning in code tracing. This sets the base for investigating RQ3. Next, RQ2 investigates whether students' performance data from T3 supports the existence of integrative skills apart from component skills, which helps examine the robustness of our prior finding, that is, the mental existence of latent integrative skills (Huang, 2018), in a new student population. This also sets the base for investigating RQ3. Finally, RQ3 investigates whether an *integration-focused instructional design* that provides deliberate practice and focused practice on integrative skills yields learning benefits beyond a design without such features. Altogether, besides demonstrating the effectiveness of the ITS technology in programming education as one contribution, a core contribution of the current work involves demonstrating the value and providing implications of integration-focused instructional design in learning technologies in general.

The article is structured as follows. Section 2 reviews related literature. Section 3 introduces our code tracing tutor T3, including the basic design and the integration-focused design. Section 4 provides details about our classroom study. Section 5 reports the results of the study centring on our research questions. At the end, in Section 6, we summarize and discuss our main results.

2 | RELATED WORK

2.1 | Domain modelling in programming learning technologies

In recent years, programming skills have become increasingly important in STEM education, yet introductory programming courses have long been regarded as challenging to students (Guzdial, 2015). Although there is abundant educational research documenting common difficulties and misconceptions in novice programmers (c.f., Qian & Lehman, 2017), it is still an ongoing challenge for the computer science education community to construct good domain models that both address the integrative nature of programming skills and maintain ease of development. Moreover, there is a lack of empirical studies investigating whether an integration-level domain model (i.e., a domain model that captures the integrative nature of programming skills) in programming learning technologies leads to improved instruction and learning compared to alternative domain models. Empirical evaluations, often referred to as 'closing the loop', have received increasing attention in mathematics learning technologies (Huang et al., 2021; Liu & Koedinger, 2017). Below, we review the existing domain modelling approaches and evaluations in programming learning technologies.

Many recent efforts have represented programming skills as programming constructs (e.g., a *for* statement), also referred to as *component-level* domain modelling in the current work, with the advantage of the ease of development (Berges & Hubwieser, 2015; González-Brenes et al., 2014; Hooshyar et al., 2015; Vesin et al., 2012; Wang et al., 2017). This approach makes it straightforward to apply automatic concept or skill extraction when building the domain model. For example, Rivers et al. (2016) used node types from Abstract Syntax Trees (ASTs)[2] produced by code parsers automatically; Hosseini and Brusilovsky (2013) used a combination of AST nodes and domain ontologies. However, Rivers et al. (2016) pointed out that the component-level representation is deficient because students' understanding of programming constructs is highly contextualized. For example, they found that errors on for loops actually increased over the course of the semester because they were used in new contexts with other constructs. Systems designed based on component-level representations likely lead to shallow learning, because a student considered as having mastered a skill may still perform poorly when the skill is used with other skills in new ways.

[2]An AST is a tree-based structure that represents the syntactic structure (i.e., the essential structural information) of a program. Each node in an AST represents a syntactic unit such as a variable, an operation or a logical operator. The children of the node represent the lower-level units associated with the current one.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

On the other hand, some early programming tutors have explored domain modelling on the level of integrative skills by using relatively complex production rules or algorithmic patterns (Anderson et al., 1989; Spohrer, 1992; Weber, 1996), and provided initial evidence of the effectiveness of such integration-level representations in supporting learning. For example, Weber (1996) represented programming skills as algorithmic patterns stored in a hierarchy of episodic frames and associated concepts. This 'episodic' learner model has proven effective for the adaptive recommendation of programming examples in ELM-ART (Weber & Brusilovsky, 2001). However, these early representations have never been expanded or ported to other systems or languages, due to the high demand involved in their knowledge engineering. Only recently have there been advances in using automatic methods to construct complex representations of programs that have the potential to represent integrative skills. For example, Akram et al. (2020) and Kovalenko et al. (2019) represented a program using features or paths each of which is the combination of a number of nearby nodes with specific relations in an AST. However, such work typically examined prediction performance in specific prediction tasks of models constructed based on the representations, rather than the effect on students' learning.

Two prior efforts have represented integrative skills using pairs of component skills (i.e., programming constructs), a simple variation to the above approaches, and have provided initial evidence of positive effects on learning. In Brusilovsky (1992), multiple programming concept pairs were connected by 'usage' links in the domain model of their ITEM/IP system, where a usage link indicated that one concept could be used in the context of another. For example, a specific kind of condition could be connected by one usage link with a loop and by another with a conditional operator. They conducted a qualitative evaluation and found that students using ITEM/IP had higher interest and performance and lower need for teachers' help to understand their bugs, compared to those using a traditional approach. In another study, Huang et al. (2017) represented integrative skills in Java code tracing as pairs of constructs (e.g., 'for & +=' denoting the use of an addition assignment statement inside a *for* statement to compute the sum of numbers) identified by experts assisted by an automatic Java parser (Hosseini & Brusilovsky, 2013). They compared such an integration-level domain model with a component-level domain model (e.g., 'for' denoting the use of a *for* statement across all contexts) by constructing learner models based on different domain models. They found that the integration-level model led to significantly better prediction of students' performance, parameter plausibility, and expected instructional effectiveness, as evaluated using the log data of a code tracing learning system; it also led to more helpful recommendations ranked by students in a lab study.

Building on the initial positive evidence of integration-level domain models in supporting learning, the current work furthers the investigation and evaluation of integration-level domain models in supporting learning to address the deficiency of component-level domain models. Specifically, we evaluated an integration-focused instructional design driven by a paired-based integration-level domain model in supporting learning of code tracing. The current work builds on our prior study of students' errors committed when integrating component skills in Python code tracing (Huang, 2018). Different from prior work on skill integration in programming that mostly emphasized procedural or cognitive load demands (du Boulay, 1986; Spohrer & Soloway, 1985), our error analysis (Huang, 2018) indicates that integration involves new skills that require knowing how component skills work together, in addition to cognitive load demands. In the current work, we represented the key integrative skills in the domain model and derived an instructional design targeting both kinds of demands.

2.2 | Code tracing skills and relevant learning technologies

Code tracing involves stepping through a program to predict changes of values of variables and outputs (Nelson, 2021). It helps students build a correct mental model of how a computer traces instructions, understand the language syntax and program semantics (Kumar, 2015). Prior work suggested that code tracing is a prerequisite for code writing (Lopez et al., 2008) and transfers to code writing (Kumar, 2015). Research also showed that introductory programming courses organized around code tracing led to significant learning outcomes (Hertz & Jump, 2013). It has been common practice for introductory programming courses to include code tracing in lectures or exercises (Hertz & Jump, 2013; Nelson, 2021; Stegeman, 2019). Meanwhile, code tracing has two features that ease the development of supporting learning technologies: (1) a new worked example or problem can be created by directly taking code from any program and running it to get the values of variables or the program outputs; and (2) there is usually a single correct answer for a code tracing question since the program is given. Thus, it is relatively easy to create a large number of code tracing problems for any programming language.

A sizable body of learning technologies has been developed to support learning code tracing. The leading approach is arguably *program visualization*, a visual step-by-step execution of programs (Jin, 2020; Sorva et al., 2013). Numerous program visualization systems were developed and evaluated over the last 20 years, yet empirical studies of their effect on learning have mixed results. Some studies reported significant improvements of learning outcomes from the use of a program visualization tool than learning without such a tool (Čisar et al., 2011), while others did not find any significant differences between them (Rajala et al., 2008). The low impact of visualization is usually attributed to its passive non-engaging nature (Naps et al., 2005). For example, Kaila et al. (2009) found that their program visualization system was useful for novices *only* if it was used in ways that required higher levels of engagement (e.g., requiring students to actively respond to questions during the visualization).

Another stream of research has focused on *automatic assessment* tools where students engage in problem solving, enter answers, and the answers are automatically graded. Although they allow for a more active learning process, *learning-by-doing*, such tools often only offer support at a coarse-grained *problem level* whereby students get feedback relative to the final output or final value of a variable of a code-

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

tracing problem (Hsiao et al., 2010; Shah et al., 2021; Thomas et al., 2019) or the final submission of code (Ihantola et al., 2010), limiting their effectiveness for supporting learning. Anderson et al. (1989), in their experiments with the LISP programming tutor, found that students who received step-level support (i.e., error feedback and hints) learned more than students with program solution evaluation only at the end of each problem, and they did so in one third the time. Without step-level guidance, novices may struggle with reaching the correct solutions, and may even disengage with learning.

An increasing number of researchers have started to study and develop code tracing technologies that offer both active learning-by-doing activities and step-level support to address the shortcomings of the above technologies. Some evidence has been found regarding the effectiveness of such technologies in supporting learning code tracing (Höppner, 2019; Jennings & Muldner, 2020, 2021; Kumar, 2020; Nelson, 2021; Qi & Fossati, 2020). These technologies can be considered as simplified instances of intelligent tutoring systems (ITS) that have been demonstrated to improve student learning in various STEM domains such as mathematics (Huang et al., 2021; Ritter et al., 2007) and physics (VanLehn et al., 2005). These empirically validated ITSs typically afford two forms of adaptivity (Aleven, McLaughlin, et al., 2016): *step-loop adaptivity*, that is, learning-by-doing support during problem solving by giving step-by-step feedback or hints personalized to each student's needs, and *task-loop adaptivity*, that is, selecting or recommending the most beneficial next activity to a learner based on their skill levels estimated by a learner model. However, many code tracing learning systems either lack task-loop adaptivity for different students (Höppner, 2019; Jennings & Muldner, 2020, 2021; Nelson, 2021), or afford limited step-loop adaptivity where no hints are provided (Jennings & Muldner, 2021) or explanations are provided after completing a problem (Kumar, 2020). Existing code tracing learning systems also vary interaction granularities (i.e., the number of intermediate steps that are required to fill in), and the most straightforward design compatible with ITS principles is to let students fill in a trace table recording the values of variables and printed outputs after the step-by-step execution of a piece of code (Kakeshita & Ohta, 2019; Risha et al., 2021; Risha & Brusilovsky, 2020). To sum up, none of the prior work has implemented and investigated a full-blown ITS technology, which affords *both* step-loop adaptivity and task-loop adaptivity, for teaching code tracing, which has the potential to promote highly effective learning.

The current work investigates the effectiveness of the instructional features provided by ITSs, specifically, the combination of active learning, step-level support, and individualized problem selection, for assisting learning in code tracing. Following the Adaptivity Grid framework (Aleven, McLaughlin, et al., 2016) which organizes research results in adaptivity into three forms of adaptivity (i.e., step-loop, task-loop, and design-loop adaptivity), the current work also implements *design-loop adaptivity*, where system designers use data about the learners in the task domain to create a new version of the system that is better adapted to common difficulties learners encounter. To start this design loop, we identified integrative skills novices struggled with in code tracing through analysis of students' errors in our prior work (Huang, 2018). In the current work, continuing this loop, we augmented a component-level domain model with integrative skills and conducted instructional design that attends to difficulties in skill integration. To close the loop, we conducted a classroom study to assess the value of the instructional features afforded by ITSs and the value of integration-focused instructional design, as well as connected the evaluation results with the original assumptions about skill integration.

3 | THE CODE TRACING INTELLIGENT TUTORING SYSTEM (T3)

Our Trace Table Tutor (T3) is an ITS for learning Python code tracing and can attend to students' difficulties in skill integration. T3 contains instructional features of an ITS, particularly active learning, step-by-step support, and individualized problem selection, which are realized by the interface, domain model, learner model, and pedagogical model (for problem selection). T3 covers multiple topics (e.g., *for*, *lists*) and has 66 problems, with 64% requiring integrative skills. T3 has two versions, the basic T3 (tutor) and the integrative T3 (tutor), explained as follows.

3.1 | The basic T3 tutor

The interface of T3 shared by both versions contains a program, a hint box, instructional text, and a trace table (Figures 1 and 2). Students need to fill in the table after each program line's execution. At each step (i.e., a table cell), students receive immediate correctness feedback through colours (red or green). Students can also ask for hints at each step, and the last hint provides the correct answer. This interface design was tested in a pilot study where we interviewed nine undergraduates taking introductory Python programming courses.

The basic T3 uses a component-level domain model with only component skill labels: // (*floor division*), % (*modulo*), += (*compound assignment*[3]), *if*, *for*, *a[i]* (*list*) (see Appendix A, Table A1 for their definitions and exemplar code). Its underlying hypothesis is a *component compositionality hypothesis* that students do not need to acquire additional integrative skills beyond component skills, and they may benefit from some practice on problems involving integration. This hypothesis is a reasonable one to be considered in a basic design of T3, because it underlies a large number of learning technologies that also follow a component-focused instructional design. In the backend, a Bayesian network-based learner model constructed based on the domain model constantly updates estimates of students' skill levels (i.e., probabilities of knowing a skill) using a previously validated formulation (Huang, 2018). The pedagogical model prioritizes the strongest unmastered skill for a student (i.e., the one among the unmastered skills that is closest to being mastered) and the problem with highest focus on this skill (i.e., having fewest other skills); when all of the student's skill levels have reached mastery (i.e., the probability of each skill exceeds 0.95), the system stops practice for the student. This

[3]There were other kinds of compound assignment in T3, such as *=. We used the label += for simplicity.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

```
1 for i in range(1, 3):
2     print(i, end='')
```

Fill in the trace table by walking through the program line by line. Only fill in a cell for (a) a line number, (b) a variable/a list element/a list that appears for the 1st time or has changed its value, (c) a specified expression or while/if condition that appears for the 1st time or has changed its evaluated value, (d) the accumulated printed output.

line	i	output

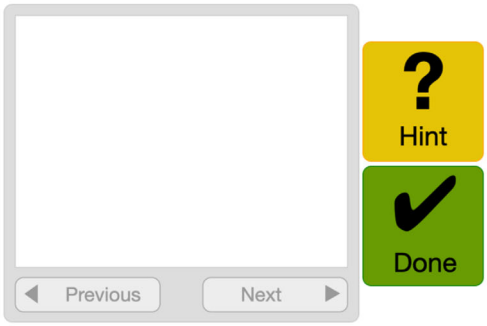


FIGURE 1 A basic problem in T3 requiring the component skill 'for'

```
1 a = [3, 2, 7, 5]
2 i = 2
3 print(a[i], end='')
4 print(a[i+1], end='')
```

Fill in the trace table by walking through the program line by line. Only fill in a cell for (a) a line number, (b) a variable/a list element/a list that appears for the 1st time or has changed its value, (c) a specified expression or while/if condition that appears for the 1st time or has changed its evaluated value, (d) the accumulated printed output.

line	i	a[i]	i+1	a[i+1]	a	output

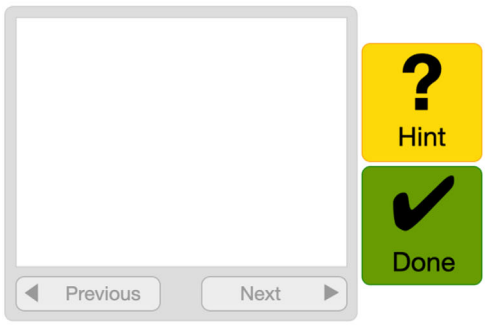


FIGURE 2 A basic problem in T3 requiring the component skill 'a[i]'

selection mechanism builds on the widely-adopted cognitive mastery learning-based problem selection in ITSs (Corbett & Anderson, 1994). Details can be found in Appendix B.

3.2 | The integrative T3 tutor

The integrative T3 optimizes the basic T3 using an integration-focused instructional design driven by an integration-level domain model. The domain model of the integrative T3 contains both component skills and integrative skills. An integrative skill is represented as a combination of component skills with certain structural constraints (e.g., a nesting relation). For example, 'for & a[i]' denotes the combination of 'for' and 'a[i]' component skills where 'a[i]' is inside a 'for' loop and one has to access an updated list element from a previous iteration to conduct operations in a new iteration. There are five integrative skills (see Appendix A, Table A2 for definitions and exemplar code) in addition to the six component skills in the integrative T3. The

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

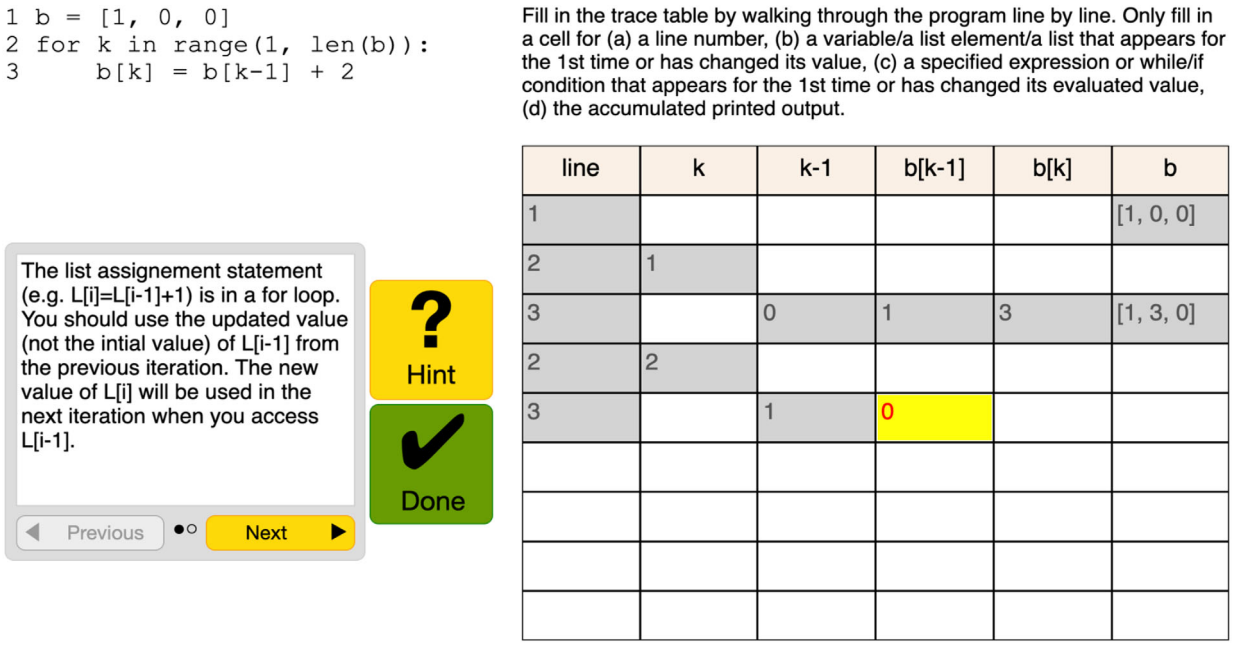


FIGURE 3 A focused integration problem in T3 requiring the integrative skill 'for & a[i]'. It only requires filling in the cells corresponding to the integrative skill (e.g., the yellow cell but not the grey cells). The yellow cell corresponds to the integrative skill 'for & a[i]' where one has to access the updated value of the list element from a previous iteration (3) in a new iteration rather than using the initial value (0). The displayed hint is used in the integrative T3

underlying hypothesis of the integrative T3 is an *integrative skill hypothesis* that students need to acquire additional integrative skills beyond component skills, and they need deliberate practice and focused practice on integrative skills. The identified integrative skills and the integrative skill hypothesis were supported by our prior study (Huang, 2018) where we designed quizzes with matched integration and decomposed problems, following the *Difficulty Factors Assessment* approach used in studying integrative skills in algebraic symbolization (Heffernan & Koedinger, 1997). Through analysis of student performance and errors committed in integration problems, we identified new integrative skills that require knowing how component skills work together, in addition to cognitive load demands. For example, we found that although many students knew 'a[i]' (i.e., access list elements from a list that has been initialized but has not been updated) and 'for' (i.e., print numbers with a for loop where later iterations do not require values from earlier iterations), they did not know 'for & a[i]' and used the initial rather than the updated value of the list element in new iterations. This specific integrative skill is related to a more general misconception in skill integration, *maintaining values across iterations*, that is, always using the initial value of the accumulator variable in each loop iteration rather than the updated value from a previous iteration identified in our prior study (Huang, 2018). Meanwhile, the integrative T3 shares the same learner modelling mechanism and problem selection mechanism as the basic T3 (as described in Section 3.1).

Both versions of T3 contain three types of problems and the same problem set: A *basic* problem not involving skill integration (Figures 1 and 2 above), a *focused integration* problem only requiring filling in the steps of integrative skills with other steps automatically filled in (Figure 3), and a *full integration* problem requiring filling in all steps for both kinds of skills (Figure 4).[4] In particular, the design of focused integration problems applied one proven instructional design method, Focused Practice Task Design (Huang et al., 2021) explained before. We created five focused integration problems each of which isolates an integrative skill and reduces cognitive load demands, drawing students' attention to the key properties of integrative skills.

Both versions of T3 differ in the step-to-skill mapping and thus also have differences in hints. In the basic T3, all steps are labelled with component skill labels and associated with hints explaining component skills. In the integrative T3, the basic steps are labelled with component skills and have the same hints as the basic T3, but the integrative steps are labelled with integrative skills and associated with hints explaining how component skills work together (Figure 3). We also ensured that the length of a hint in the integrative T3 was similar to the length of the hint of the same problem step in the basic T3 (see Appendix A, Figure A1 for a comparison).

The differences in the domain models between the two versions of T3 result in their differences in problem distribution and sequencing, according to the shared problem selection mechanism (as described in Section 3.1). First, the integrative T3 provides integration problems as the main problems, provides focused integration problems as the main integration problems, and has more such problems than the basic T3. This is because the targeted population of T3 is students who already have a good level of component skills after initial lectures and

[4]This problem requires another integrative skill 'a[i] & =', denoting the integration of a list and assignment where one has to write down the updated list with all its elements. For simplicity of illustration, we did not mention it in the caption.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

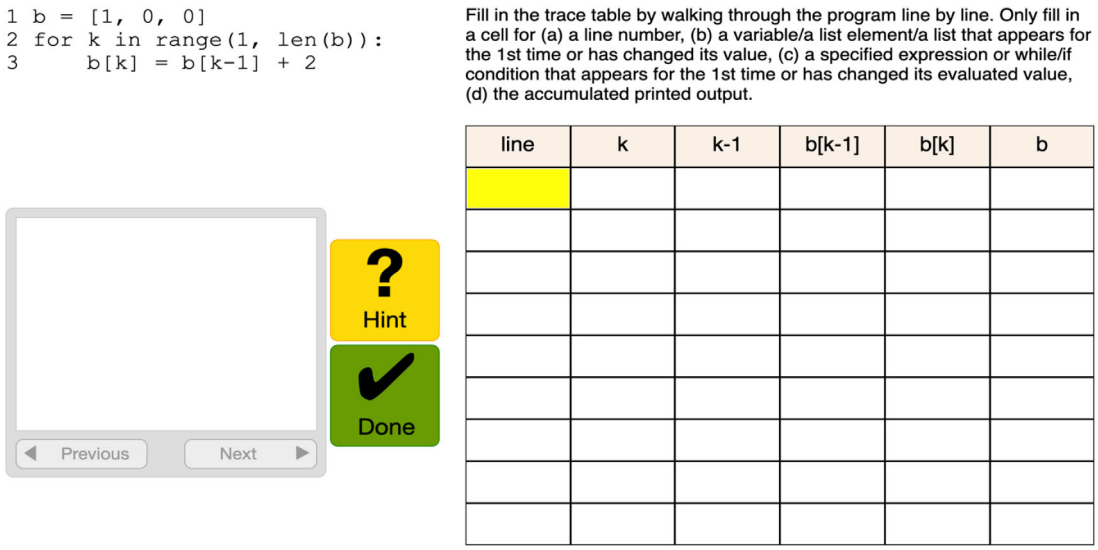


FIGURE 4 A full integration problem in T3 requiring component skills 'for', 'a[i]' and the integrative skill 'for & a[i]'

exercises but struggle with integration in more complex problems. The integrative T3 labels integration problems with integrative skills not used for labelling basic problems and tracks integrative skill levels separately allowing such students to receive just what they need (i.e., integration problems), whereas the basic T3 labels both basic and integration problems with only component skill labels resulting in a high proportion of basic problems. Second, the integrative T3 prioritizes practice on basic problems before (rather than after) integration problems. This is because under the problem selection mechanism, if students have not yet mastered component skills,[5] the integrative T3 prioritizes component skills and basic problems; if a student has mastered component skills, it avoids selecting component skills and thus basic problems because only unmastered skills are considered. In comparison, the basic T3 interleaves the basic problems and integration problems. It may select integration problems before a student has mastered component skills and may still select basic problems after a student has mastered component skills. This is because each skill label is associated with both basic and integration problems, there is no guarantee of progressive problem selection between the two types.

Figure 5 shows the logic model contrasting the two versions of T3 through a hypothesized causal chain from the domain model and underlying hypothesis on skills and instruction, to instructional design features, and finally to outcomes. We will explain the outcome measures in Sections 4 and 5. We will report on the overall outcome comparison between the two versions of T3 and mediating effects through specific instructional features towards supporting the causal chain in Section 5.

To build T3, we utilized the Cognitive Tutor Authoring Tools (CTAT),[6] a widely-used tool to create ITSs (Aleven, McLaren, et al., 2016; Aleven, McLaughlin, et al., 2016). We implemented an infrastructure that can automatically generate CTAT required step-by-step solution graphs (and HTML files) with skill labels and hints embedded for a given Python program. To host T3, we used the learning management system TutorShop,[7] which calls our customized learner modelling and problem selection service. To log the data, we used the service provided by DataShop.[8] Details could be found in Huang (2018).

4 | STUDY DESIGN AND DATA COLLECTION

4.1 | Participants

We conducted a classroom study with 74 students in an introductory Python programming course at a university in [a Spanish-speaking country masked for review]. All students were first- or second-year engineering school students, and 85% were male.

4.2 | Procedures

The study was held as an in-class practice session 2 weeks after the lecture on the topic of *lists* (other topics involved in T3 had been covered earlier) and students had already done some basic exercises on the *lists* topic. The session lasted for about 80 min and involved several tasks: demo, pre-assessment, practice, and post-assessment. Students were randomly assigned to one of the two versions: 36 used the basic T3 (the control condition) and 38 used the integrative T3 (the experimental condition). All hints and instructions were provided in Spanish.

[5] In this paragraph, having mastered component skills means students are estimated to master *all* component skills according to an *accurate* domain and learner model.
[6] https://github.com/CMUCTAT/CTAT/wiki.
[7] https://tutorshop.web.cmu.edu/.
[8] https://pslcdatashop.web.cmu.edu/.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

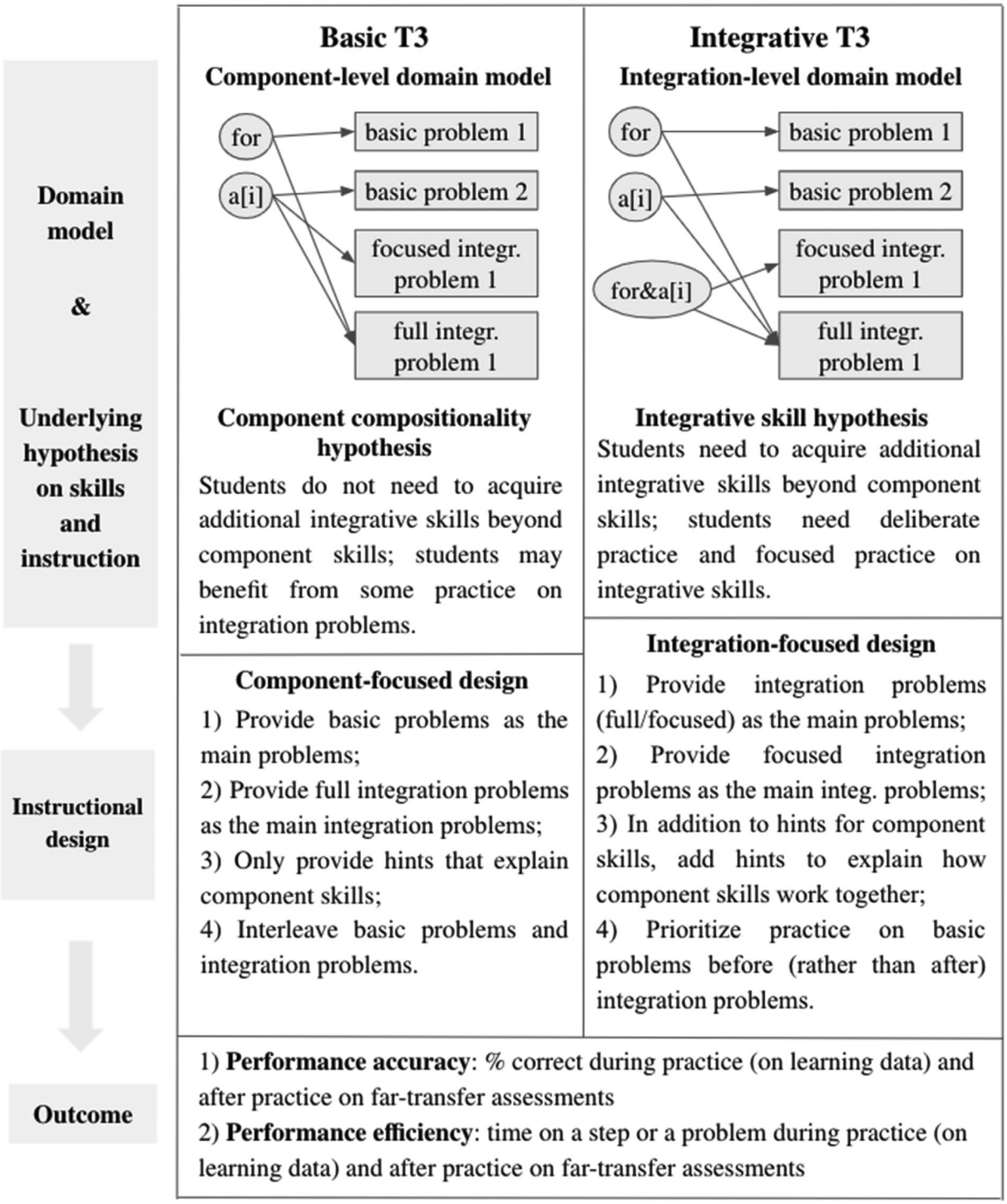


FIGURE 5 The logic model contrasting the two versions of T3 through a hypothesized causal chain

4.3 | Measures

The pre-assessment consisted of six basic problems, each of which targeted one component skill.[9] The post-assessment included the same six basic problems with changes that did not affect the involved skills (e.g., the values of the literals) and five integration problems covering all the five targeted integrative skills. Integration problems were not included in the pre-assessment to reserve more time for practice, since we could not limit time on the pre-assessment in the experimental platform. Each problem displayed a small program and asked students to write down the printed output or the final value of the key

[9]The last basic problem aimed at assessing students' levels on '%', but it also involved 'if'.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

variable; both assessments were computer based and offered no trace tables (see Figure A2 in Appendix A for the interface). Each problem was graded as correct or incorrect on the last attempt. We refer to the proportion correct on the pre- or post-assessment as pre- or post-assessment *score* for a student. This list of assessment problems could be found in Appendix A, Tables A3 and A4. The assessment problems allow us to study *far transfer* learning since they did not provide trace table support and involved more integrative skills or component skills in a single problem than most practice problems,[10] and to examine the overall problem level performance.

In addition to the assessment data, we also collected and analysed student learning data as they practiced with T3. We used only the first attempts of problem steps for analyses following common practice in analysing ITS log data. The learning data provides a relatively larger amount of data per student, provides insights into step-level performance, and allows us to study *near transfer* learning as we examine student performance changes during instruction with trace table support.

5 | RESULTS

This section presents three sets of analyses to answer our three research questions. For statistical analysis, we used two-sided t-tests (after confirming approximate normality and equal variance) or Wilcoxon signed-rank tests; we also used multiple regression and mixed-effects modelling. Cohen's *d* is reported for effect sizes. Whenever the condition variable was used in a regression model, we coded 0 for the basic T3 condition and 1 for the integrative T3 condition.

5.1 | RQ1: Student learning in the code tracing ITS (T3)

RQ1 investigates whether the instructional features of a full-scale ITS, T3, produce effective learning in code tracing. We examined learning gains on data from both conditions.

First, we examined learning gains using assessment data by the differences between post- and pre-assessment scores of students. We only considered basic problems which were the only overlapping problems in both assessments. The difference ($M = 0.11$, $SD = 0.19$) between post-assessment scores ($M = 0.89$, $SD = 0.15$) and pre-assessment scores ($M = 0.78$, $SD = 0.20$) was statistically significant with a medium effect size [$t(73) = 4.74$, $p < 0.001$, $d = 0.55$].

Then, we examined student learning on both basic and integrative skills using the learning data starting with visual inspection of learning curves. Learning curves were often used as a subtle way to measure learning outcomes in adaptive educational systems (Martin et al., 2011). A *learning curve* depicts the average error rates of students over successive practice opportunities (i.e., first attempts of problem steps) for one skill or aggregated over a group of skills. The learning curves for component skills (Figure 6a) and integrative skills (Figure 6b) show a decreasing error rate that indicates students were learning throughout use of T3. As shown in Figure 6a (the blue curve which is a smoothed curve), students had an around 30% average error rate (i.e., 70% correct) on their first opportunity on basic problems and that average error rate reduced to 10–20% as students experienced more opportunities to learn from practice and feedback (e.g., reaching less than 20% after 10 opportunities). As shown in Figure 6b (blue curve), students had a much higher error rate on integration problems, starting at about a 55% average error rate on the first opportunity, and gradually transitioned to an error rate around 25% after 15 or so opportunities.

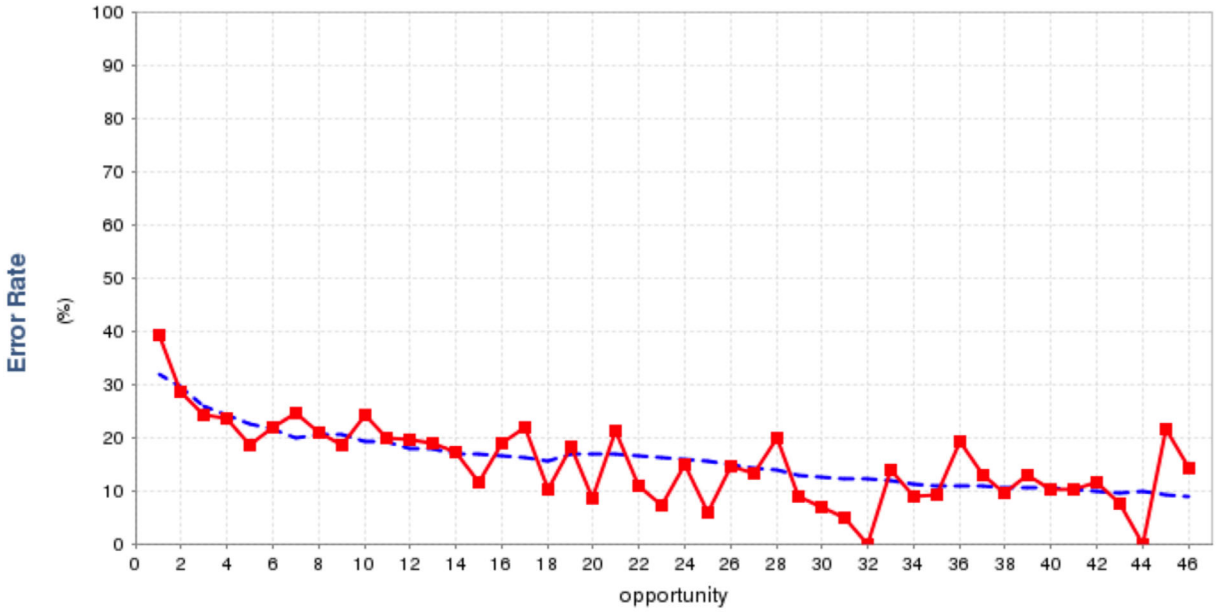
We further utilized the Additive Factor Model (AFM; Koedinger, McLaughlin, & Stamper, 2012), a learner modelling technique often used in conjunction with learning curves (e.g., for plotting the blue predictive curves in Figure 6) to rigorously examine learning in educational systems (Huang et al., 2021; Koedinger, Corbett, & Perfetti, 2012; Koedinger, McLaughlin, & Stamper, 2012). AFM is a growth model generalization of psychometric item response theory models (De Boeck & Wilson, 2004). It is a logistic regression model given a mapping between problem steps and skills. AFM predicts the probability that a student *i* will get a problem step *j* correct based on the student's proficiency (θ_i), the initial difficulty (β_k) and learning rate (γ_k) of each required skill *k* (indicated by q_{jk} with value 0 or 1) as the student gets practice opportunities on this skill (T_{ik}). We used the DataShop (Koedinger et al., 2010) implementation of AFM, which introduces the by-student random intercepts.

$$\ln\left(\frac{p_{ij}}{1-p_{ij}}\right) = \theta + \sum_{k=1}^{K} q_{jk}\beta_k + \sum_{k=1}^{K} q_{jk}\gamma_k T_{ik}. \quad (1)$$

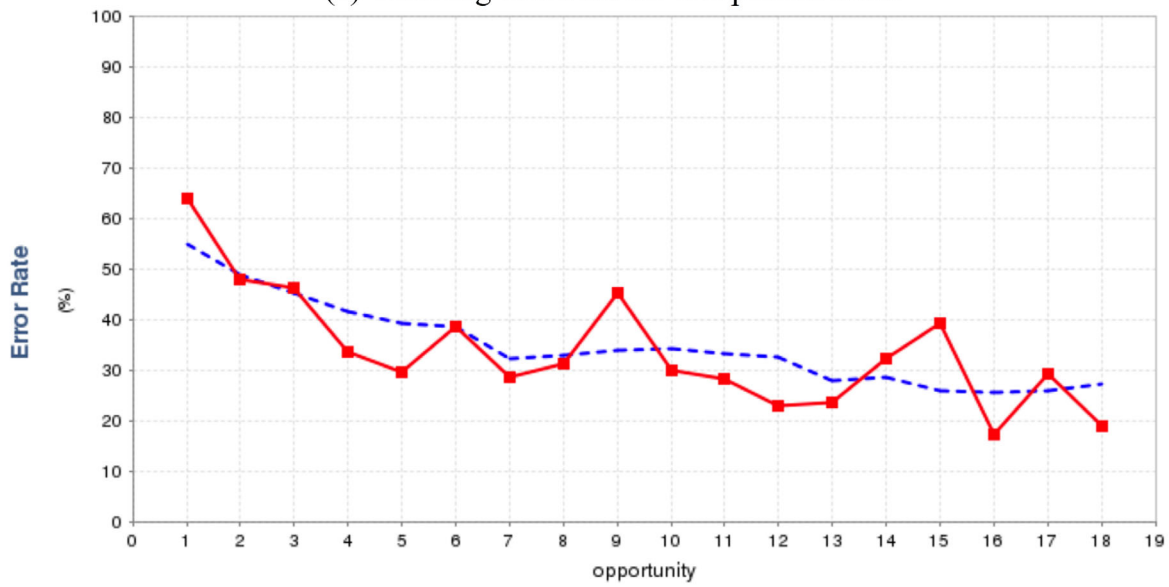
We evaluated whether learning occurred by inspecting the skill learning rate parameters (γ_k). The average learning rates of the component skills, integrative skills, and overall skills are 0.13, 0.25 and 0.18 respectively. The positive learning rates indicate that students increased their performance accuracy (correctness) as they got more practice. Based on the fitted parameters, we further estimated the learning gain using the fine-grained learning data when students practiced with trace table support: we computed the difference in the probabilities of succeeding between the last and the first practice opportunity of a given skill in the entire study span for each student. We found significant learning gains (i.e., differences) with large effect sizes for component skills, integrative skills, and overall skills [basic: $t(73) = 10.15$, $p < 0.001$, $d = 1.18$, $M = 0.06$, $SD = 0.05$; integrative: $t(73) = 9.73$, $p < 0.001$, $d = 1.13$, $M = 0.11$, $SD = 0.10$; overall: $t(73) = 11.38$, $p < 0.001$, $d = 1.32$, $M = 0.08$, $SD = 0.06$].

[10]Moreover, the last integration problem (id = 11) involved reversing a list by swapping list elements that were not involved in any practice problems, which further adds to the assessment of far transfer learning in post-assessment.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License



(a) Learning curve for the component skills



(b) Learning curve for the integrative skills

FIGURE 6 The learning curves for the component skills (a) and the integrative skills (b) in T3. In both curves, opportunities with fewer than 20 students were excluded. Red and blue curves correspond to the raw and smoothed (AFM predicted) error rates respectively

5.2 | RQ2: Existence of integrative skills in code tracing

RQ2 investigates whether students' performance data from T3 supports the existence of integrative skills, that is, confirms a composition effect in code tracing, namely, that problems that involve a 'composition' of two (or more) component skills are harder than predicted by student performance on problems involving those component skills separately.

We examined the post-assessment data for evidence of a composition effect. In our past work (Huang, 2018), we investigated the existence of a composition effect in Python code tracing by comparing student performance on integration problems with matched basic problems. We followed this approach here, comparing correctness on an integration problem (*pI*) with estimated correctness from all the basic problems (*pB*) involved in that integration. For example, the first row of Table 1 shows that for the first integration problem, the average post-assessment correctness (of students) on the basic problems for component skills 'for' and 'a[i]' is 95% and 88% respectively. Being correct on both (*pB*) is estimated by the product at 84% correct. In contrast, student performance on the integration problem involving the integrative

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 1 Integrative skill levels (*pI*/*pB*) and composition effects (*pB* − *pI*), along with their statistical significance and effect sizes, computed using post-assessment performance

Integ. prob.	Component skills	Basic problem average correctness (component skill level)		Integration problem average correctness (*pI*)	Integrative skill level (*pI*/*pB*)	Composition effect statistical analysis: Is *pB* − *pI* > 0?		
		Individual	Product (*pB*)			*t*(73)	*p*	*d*
1	for, a[i]	0.95, 0.88	0.84	0.35	0.42	8.78	<0.001	1.02
2	for, a[i], +=	0.95, 0.88, 0.97	0.81	0.34	0.42	8.78	<0.001	1.02
3	for, a[i], if	0.95, 0.88, 0.91	0.76	0.41	0.54	5.75	<0.001	0.67
4	for, a[i]	0.95, 0.88	0.84	0.28	0.33	10.06	<0.001	1.17
5	for, a[i], //	0.95, 0.88, 0.88	0.74	0.35	0.48	7.08	<0.001	0.82
Average		0.89	0.80	0.35	0.44	10.66	<0.001	1.24

skill 'for & a[i]' was only 35% correct. The difficulty of the whole (*pI*) is significantly greater than the predicted difficulty of the parts (*pB*) (see last three columns in Table 1). The product of the performance on basic problems corresponds to a hypothesis that an integration problem is no more than getting all the decomposed basic problems correct (95%*88% = 84%). However, the fact that the integration problem has much lower performance (35%) than the product indicates that some extra skill is needed for students to learn to integrate component skills. We call this an integrative skill, and we can predict integration problem correctness (*pI*) as the product of integrative skill correctness (*pIS*) and combined component skill correctness (*pB*), that is, $pI = pIS^{*}pB$. We can thus estimate integrative skill correctness as $pIS = pI/pB$.[11] This value is shown in the sixth column (*pI*/*pB*) and is 42% for the first integration problem.

Across the five integration problems, students did quite well on basic problems, averaging 89% (see the last row in Table 1), and joint correctness (*pB*) is also predicted to be high, averaging 80%. However, correctness on integration problems (*pI*) is much lower, averaging 35%. Estimated integrative skill correctness (*pI*/*pB*) ranges from 33% to 54% with an average of 44%. The last three columns in Table 1 show statistical analysis of composition effects (*pB* − *pI*), that is, results of paired t-tests comparing each student's combined basic problem performance (*pB*) with their integration problem performance (*pI*). Across all integration problems this difference is highly significant ($p < 0.001$ in all cases) and highly substantial with large effect sizes in most cases ($d > 0.8$). These results confirm the substantial composition effect in code tracing and indicate that certain integrative skills have to be acquired beyond component skills.

5.3 | RQ3: Learning benefits of integration-focused instructional design

RQ3 investigates whether an integration-focused instructional design yields learning benefits beyond a design without such focus. We compared student learning in both conditions from both performance accuracy and efficiency.[12] We started with comparing the practice time to confirm that any differences in learning outcomes between the two conditions only resulted from instructional differences rather than differences in total practice time. Indeed, there was only a small and nonsignificant difference [$t(72) = -1.37$, $p = 0.17$] in total practice time (control: $M = 41.00$ min, $SD = 9.40$ min; experimental: $M = 44.10$ min, $SD = 9.70$ min).

5.3.1 | Performance accuracy comparison

We first compared the performance accuracy (score) measured by the assessment data where trace tables were not provided. We first examined basic problems, which were the only overlapping problems on pre- and post-assessments. Regarding pre-assessment levels, we found a marginally significant difference [$t(72) = 1.69$, $p = 0.096$, $d = 0.39$], with the basic T3 condition ($M = 0.82$, $SD = 0.18$) starting slightly higher than the integrative T3 condition ($M = 0.75$, $SD = 0.21$).[13] To control for pre-assessment differences, we ran a regression analysis predicting the post-assessment score per student given the pre-assessment score and the condition indicator on basic problems. The integrative T3 condition yielded higher post-assessment scores, yet the difference was not statistically significant ($b = 0.01$, $p = 0.77$). We also did not find an aptitude-treatment interaction when adding an interaction term between the pre-assessment score and the condition ($b = -0.01$, $p = 0.85$). In addition, for integration problems, we ran a regression analysis predicting each student's post-assessment score on integration problems given the pre-assessment score on basic problems (since integration problems were not available in the pre-assessment) and the condition indicator. There was no statistical difference between conditions ($b = 0.08$, $p = 0.36$), and we did not find an interaction between the pre-assessment score and the condition ($b = -0.08$, $p = 0.36$).

[11] In some post-assessment integration problems, multiple integrative skills were involved in a single problem. pIS thus could denote the joint probability of getting all integrative skills correct in a problem.

[12] In this section, performance is further decomposed into accuracy and efficiency; in other sections, the word *performance* by default denotes performance accuracy specifically.

[13] Although students were randomly assigned to two conditions, statistically, random assignment does not guarantee that the conditions are matched or equivalent and has a possibility to fail (Goldberg, 2019).

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 2 Descriptive statistics and statistical tests of the median time (second) of completing a problem in different problem sets between the basic T3 control (CT) condition and the integrative T3 experimental (EP) condition

Assessment type (problem type)	CT ($N = 36$)		EP ($N = 38$)		CT-EP	(CT-EP)/CT	Two-sample t-test		
	M	*SD*	*M*	*SD*	*M*	Reduct. ratio	*t*(72)	*p*	*d*
Pre-assess (basic)	72	29	67	25	5	–	0.73	0.47	–
Post-assess (basic)	38	15	31	11	7	17%	2.08	0.04	0.49
Post-assess (integ.)	97	42	77	36	20	20%	2.12	0.04	0.49
Post-assess (overall)	57	23	46	15	11	19%	2.39	0.02	0.56

We then compared the performance accuracy measured by the learning data generated from student interactions with T3 problems which all provided trace tables and we could estimate students' initial levels of integrative skills from AFM models fit to the data. We examined the difference between the final and initial estimated probabilities of success (*pS*) by AFM models fitted to the learning data of each condition. We ran three regression models predicting the average final *pS* per student given the average initial *pS* and the condition, where we computed the average final or initial *pS* per student on overall skills, component skills, and integrative skills respectively. The integrative T3 condition led to higher final *pS* than the basic T3 condition, and the condition differences were significant with medium effect sizes for overall skills ($b = 0.05$, $p < 0.001$, $d = 0.66$) and component skills ($b = 0.05$, $p = 0.008$, $d = 0.50$), and the condition difference was marginally significant for integrative skills ($b = 0.03$, $p = 0.08$, $d = 0.66$).

This set of analyses shows that the integrative T3 condition did not lead to reliably higher performance accuracy improvement measured by assessment data where trace tables were not provided and pre-assessment on integrative skills was not available. However, it led to reliably higher performance accuracy improvement for overall skills and component skills measured by the learning data where trace tables were provided and initial levels of integrative skills could be estimated and controlled for, compared with the basic T3 condition.

5.3.2 | Performance efficiency comparison

The time students spend on performing a task captures another important aspect of student performance: *efficiency*. This measurement was used in the well-studied power law for practice that depicts a phenomenon where learners' time to perform a task decreases over practice (Neves & Anderson, 1981). Here we compared time needed to complete an assessment problem and to complete a practice problem step in two conditions to see whether the integrative T3 condition led to performance efficiency benefits.

First, we compared performance efficiency on the assessment data. Since there were missing values due to some students' not completing all problems,[14] we employed two sets of analyses: t-tests on imputed data and linear mixed effect modelling (LMM) without imputing the data. We considered both incorrect and correct responses rather than using only correct responses since those sample sizes were too small. For the t-tests with data imputation, for each student, we calculated the median time over a set of problems, and then compared the average of the medians of the two conditions. For students who failed to submit all problems, the median time introduced bias towards the time required on earlier problems (note that all basic problems were positioned before integration problems), so we conducted imputation as follows: for a problem type (basic/integration) under an assessment type (pre- or post-assessment) of a condition, we filled the missing value with the average of the values from that problem type under the same assessment type and condition, after conducting a 90% winsorization if the distribution failed the normality test. Table 2 reports the results. There was no statistical difference by condition in the pre-assessment ($p = 0.47$). However, students in the integrative T3 condition spent significantly less time on post-assessment basic problems ($p = 0.04$, 17% less), integration problems ($p = 0.04$, 20% less), and all problems ($p = 0.02$, 19% less) with approximately medium effect sizes.

In the LMM analysis without imputation, we followed the common practice of logarithmic transformation of time since the time distribution was highly skewed. The LMM predicts the log-transformed time on each submitted post-assessment problem of a student, based on the condition indicator, the log-transformed total time on pre-assessment (the coefficient of which was significant in our model), the problem indicator (random factor), and the student indicator (random factor). There was a significant difference between the conditions ($b = -0.15$, $SE = 0.07$, $X^2(1) = 4.08$, $p = 0.04$), and students in the integrative T3 condition spent 15 seconds less per problem on average. Considering that there was no statistical difference in performance accuracy in post-assessment, these results show that the integrative T3 condition led to significantly higher performance efficiency in post-assessment compared to the basic T3 condition.

Second, we examined performance efficiency within the learning data. To reliably measure the decrease of time spent on an opportunity (i.e., a problem step) over practice opportunities, we adapted AFM to predict the time spent on each opportunity, using the same predictors and domain model used in Section 5.1 Equation (1) and only changed the dependent variable. Again, we applied a logarithmic transformation to the time dependent variable when fitting the model due to its skewed distribution and converted to the original scale in follow-up analyses. We considered all attempts and only correct

[14] In the basic T3 and integrative T3 conditions, the proportion of students that submitted all 11 post-assessment problems were 75% (27/36) and 92% (35/38) respectively.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE 3 The number and proportion of problems selected in the two conditions. The mean (*SD*) over students, overall increase (inc.) ratios and two-sample *t*-test significance are reported

	# of practice problems of a problem type				Prop. of a problem type	
Problem type	**Basic T3**	**Integ. T3**	**Difference**	**Inc. ratio**	**Basic T3**	**Integ. T3**
Basic problem	6.8 (5.6)	2.9 (2.5)	−3.9***	−57%	0.57 (0.22)	0.21 (0.13)
Integ. problem	6.1 (6.6)	10.3 (6.9)	4.2*	69%	0.43 (0.22)	0.79 (0.13)
Focused integ. problem	1.1 (1.3)	3.4 (1.4)	2.3***	209%	0.08 (0.14)	0.34 (0.21)
Full integ. problem	5.1 (5.7)	6.8 (6.2)	1.7	33%	0.34 (0.21)	0.44 (0.18)

†$p < 0.10$; * $p < 0.05$; ** $p < 0.01$; *** $p < 0.001$.

attempts.[15] We first checked differences in the average estimated time on first opportunities and found that on all attempts, there was a significant difference between the conditions on component skills [$t(72) = 3.15$, $p = 0.002$][16] (where the basic T3 condition had a smaller mean), but not on integrative skills [$t(72) = 0.57$, $p = 0.57$], nor on overall skills [$t(72) = 1.48$, $p = 0.14$]. We then ran regression models predicting the average estimated time on last opportunities given the condition indicator and the average estimated time on first opportunities over a set of skills for each student. The integrative T3 condition led to less time on last opportunities and the differences between conditions were significant with large effect sizes for basic, integrative, or overall skills (basic: $b = -0.40$, $p < 0.001$, $d = 0.80$; integrative: $b = -2.19$, $p < 0.001$, $d = 1.39$; overall: $b = -1.25$, $p < 0.001$, $d = 1.38$). The same results hold when we consider only correct attempts (basic: $b = -0.28$, $p < 0.001$, $d = 0.78$; integrative: $b = -1.21$, $p < 0.001$, $d = 1.16$; overall: $b = -0.71$, $p < 0.001$, $d = 1.10$). Considering that the integrative T3 condition led to higher performance accuracy improvement during practice, these results show that the integrative T3 condition led to greater performance efficiency gains during practice than the basic T3 condition.

5.3.3 | Mediation analysis

In this section, we conducted a preliminary mediation analysis to support the effectiveness of specific instructional design features on the overall intervention effects. Among the outcome variables covered above, we picked the one where we observed the largest effect sizes: the performance efficiency variable on the learning data measured by the average estimated time on last opportunities of skills. Among the potential mediators, we only considered those for which we could obtain measures with no or less ambiguous interpretations of the effectiveness of a design.[17] We conducted the mediation analysis separately for different skill types since different design features may impact differently for component skills and integrative skills.

We investigated three hypothesized mediators: the number of integration problems, the number of focused integration problems, and a problem sequencing feature (explained below). The hypothesized mediating processes are as follows. The integrative T3 condition should lead to more integration problems and more focused integration problems (as explained in Section 3.2) which should lead to greater overall efficiency improvement on integrative skills than the basic T3 condition. Meanwhile, although the integrative T3 condition was not expected to yield more basic problems, more integration problems may also yield greater overall efficiency improvement on component skills than the basic T3 condition, because integration problems also contain steps of component skills and practice on integrative skills may deepen the learning on component skills through learning when and how to apply a specific type of skill. The integrative T3 condition may also select integration problems after (rather than before) basic problems or avoid selecting basic problems after integration problems to a greater degree (as explained in Section 3.2) which may yield greater overall efficiency improvement for both basic and integrative skills than the basic T3 condition, because the selected problems may be neither too difficult nor too easy for students. We report descriptive statistics and statistical tests to first confirm that the values of the mediators are different between two conditions, and then report mediation analysis results.

We first compared the number and proportion of problems of different problem types between conditions. As shown in Table 3, the integrative T3 had a greater focus on integration problems than the basic T3: it led to a significantly higher number of integration problems and a much higher proportion of integration problems (79% vs. 43%). The integrative T3 also selected significantly more focused integration problems and a much higher proportion of the selected to the available focused integration problems (3.4/5 = 68% vs. 1.1/5 = 22%). Next, we compared problem sequencing features between conditions. We defined four types of transitions for each consecutive problem pair and computed the proportion of each type for each student. Among the four types, the integration-to-basic transition is the type that is most likely to indicate suboptimal problem selection. We found that the integrative T3 condition provided a significantly lower proportion of integration-to-basic transition, reducing

[15]There was sufficient data considering only correct attempts (>2000 transactions and 32 students per condition).

[16]Although there was no statistical difference between pre-assessment time between conditions, it is possible that there was a statistical difference in the time on first opportunities between conditions, because both conditions differed in when and which first basic problems were given to students.

[17]An example of a potential mediator for which measures may lead to ambiguous interpretation is the usage of hints, because higher usage may indicate higher usefulness of hints from a good design, but it may also indicate higher difficulty of steps that make students request hints from a suboptimal design.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

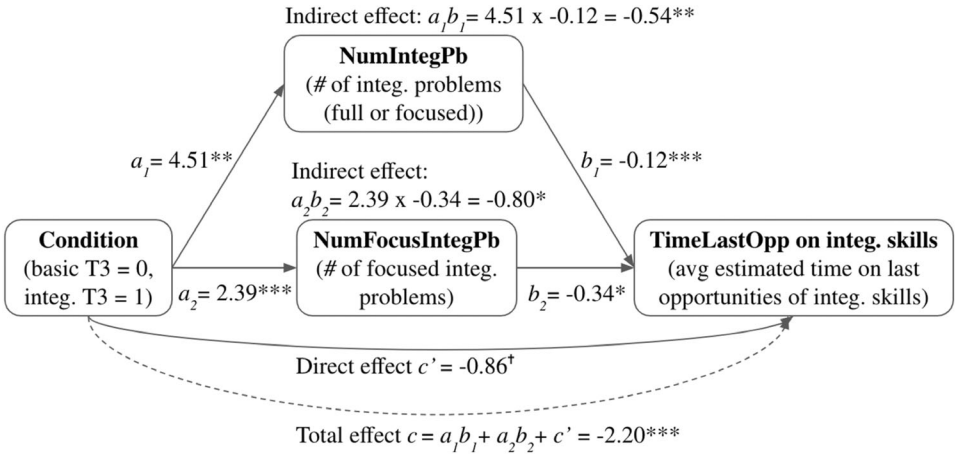


FIGURE 7 The mediation model with path unstandardized coefficients for testing the indirect effect of condition on performance efficiency (measured in the learning data) through mediators for integrative skills. (†p < 0.10, *p < 0.05, **p < 0.01, ***p < 0.001)

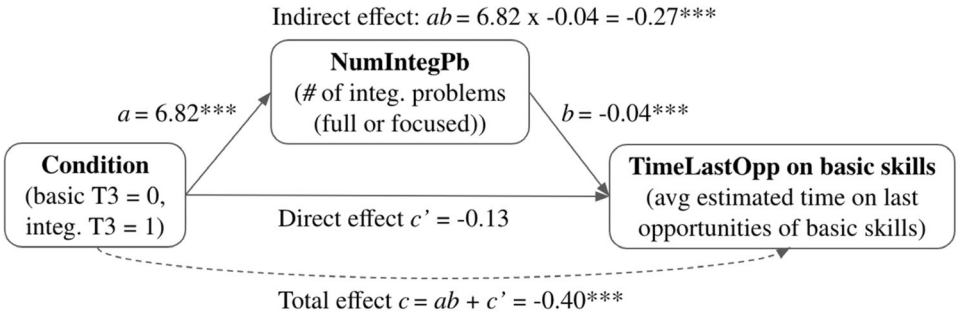


FIGURE 8 The mediation model with path unstandardized coefficients for testing the indirect effect of condition on performance efficiency (measured in the learning data) through a mediator for component skills. (†p < 0.10, *p < 0.05, **p < 0.01, ***p < 0.001)

that of the basic T3 condition by 33%. Detailed statistics can be found in Appendix C, Table C1.

We then conducted an mediation analysis (Rosseel, 2012). We tested the indirect effect of the condition (*Condition*) on the average estimated time on last opportunities of skills (*TimeLastOpp*) for students through one or a few mediators, using 1000 bootstrapped samples. The average estimated time on first opportunities of skills was included as a covariate. Both time variables were in the original rather than logarithmic scale. For integrative skills, we first only considered a single mediator. The number of integration problems (*NumIntegPb*) significantly mediated the relationship ($ab = -0.72$, 95% CI [−1.16, −0.34], $p = 0.001$), and so did the number of focused integration problems (*NumFocusIntegPb*; $ab = -1.52$, 95% CI [−2.25, −0.87], $p < 0.001$); but the average proportion of the integration-to-basic problem transition (*PropIntegToBasic*) did not mediate the relationship ($ab = -0.01$, 95% CI [−0.16, 0.17], $p = 0.95$). We then considered the two variables that showed mediating effects at the same time. As shown in Figure 7, *NumIntegPb* significantly mediated the relationship ($a_1b_1 = -0.54$, 95% CI [−0.95, −0.18], $p = 0.007$), and *NumFocusIntegPb* provided an additional significant mediation effect ($a_2b_2 = -0.80$, 95% CI [−1.61, −0.12], $p = 0.03$). Both mediators together partially to fully mediated the relationship ($c' = -0.86$, 95% CI [−1.75, 0.05], $p = 0.05$). We also confirmed that *PropIntegToBasic* did not provide an additional mediating effect in a three-mediator model ($ab = 0.01$, 95% CI [−0.10, 0.13], $p = 0.84$). For component skills, we consider a single mediator first. As shown in Figure 8, *NumIntegPb* significantly mediated the relationship ($ab = 6.82 * (-0.04) = -0.27$, 95% CI [−0.40, −0.15], $p < 0.001$) with full mediation ($c' = -0.13$, 95% CI [−0.37, 0.05], $p = 0.25$); *PropIntegToBasic* did not mediate the relationship ($ab = -0.01$, 95% CI [−0.05, 0.03], $p = 0.62$). We also confirmed that *PropIntegToBasic* did not provide an additional mediating effect in a two-mediator model ($ab = -0.00$, 95% CI [−0.03, 0.03], $p = 0.87$).

6 | DISCUSSION AND CONCLUSIONS

The current work presents the development and empirical evaluation of instructional design targeting students' difficulties in code tracing particularly in integrating component skills in an intelligent tutoring system, Trace Table Tutor (T3). We obtained three major findings centring on three research questions (RQs). First, the instructional features of this full-scale ITS for code tracing, particularly active learning, step-level support, and individualized problem selection, produced effective learning in code tracing, as evidenced by the significant learning gains with medium to large effect sizes measured by the assessment data and learning data. Second, students' performance data collected from T3 supports the existence of integrative skills in code tracing. Learning how to trace basic programming constructs in isolation does not sufficiently prepare students to trace more complex programs where these constructs are combined; students' much lower-than-predicted performance on integration problems indicates they need to acquire new skills for integration. Third, an instructional design that provides deliberate practice and focused practice on integrative skills yields learning benefits beyond a design without such features. The integrative T3 led

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

to more efficient post-assessment performance (i.e., faster problem completion) and greater performance efficiency gains during practice (both significant with medium to large effect sizes), as well as higher performance accuracy improvement during practice (significant with medium effect sizes for overall skills and component skills), compared to the basic T3. Below, we first discuss our findings in the context of existing research and their practical implications (Sections 6.1–6.3) and highlight limitations and future work (Section 6.4) at the end.

6.1 | ITS instructional features and scaling up T3

The current work demonstrates the effectiveness of instructional features of a full-scale ITS, particularly active learning, step-level support, and individualized problem selection for learning code tracing (RQ1). The effect sizes of learning gains (medium to large) are comparable to those reported in experimenter-designed tests in the classic successful studies of ITSs in mathematics and physics (Koedinger et al., 1997; VanLehn et al., 2005). The accumulated evidence across domains shows the value of the combo of the three features supported by full-scale ITSs and suggests that enhancing existing code tracing learning technologies to have these three features may boost their effectiveness. Although the current study did not experimentally investigate the effect of each individual feature, there is accumulated empirical evidence of the effectiveness of each feature (Aleven, McLaughlin, et al., 2016; Koedinger et al., 2015). In addition, future research can investigate alternative forms to implement these features (Aleven, McLaughlin, et al., 2016).

The current T3 only involves a small set of the integrative skills in Python programming, yet it can be relatively easily scaled up to handle more complex integrative skills due to our technical infrastructure. Given a domain model and a skill-to-hint mapping, a new version of T3 can be easily created automatically under our infrastructure. The main effort lies in constructing the domain model: one needs to specify the combinations of component skills and the structural constraints as integrative skills and then these skill specifications will be implemented in a program extending our existing program to automatically annotate the steps. Section 6.2 below outlines promising methods towards this goal.

The current work may also help build technologies for learning code writing. For example, the empirical method for identifying integrative skills and the instructional design for supporting learning integrative skills can be applied to the code writing area. However, the differences between code-tracing and code-writing skills demand new investigation, since some studies raised doubts on the relation between code tracing and code writing (Denny et al., 2008), and it is also unclear whether the skill combination structures that cause difficulty in code tracing are also the ones that cause difficulty in code writing.

6.2 | Integration-level domain modelling in programming learning technologies

The current study provides support (RQ2) and shows the value of one way of integration-level domain modelling (RQ3), that is, using the *combination* of component skills, compared to the common component-level domain modelling, which adds to prior work using combination-based representations in programming learning research. This combination-based representation is founded on our prior empirical investigation of integrative skills in code tracing (Huang, 2018), and the current analysis on a new student population (RQ2) further supports the robustness of the existence of integrative skills apart from component skills. The current study extends prior works using pair-based representations for programming skills in other programming languages and learning technologies (Brusilovsky, 1992; Hsiao et al., 2010; Huang et al., 2017). Pair-based representations may be considered as the simplest form among various representations of programming skills, including generic program fragments encoding specific arrangements of constructs (typically specified by experts) in computing education research (e.g., *plans* (Soloway & Ehrlich, 1984), *schemas* (Rist, 1989) and *templates* (Clancy & Linn, 1999)), common basic *algorithms* (e.g., *Sum and Average Value*) in classic textbooks (Horstmann, 2020), as well as combinations of nodes (some of which correspond to programming constructs) from Abstract Syntax Trees (ASTs) with structural constraints through automatic AST mining (Akram et al., 2020; Kovalenko et al., 2019). Our current study adds to this field by providing empirical evaluation of combination-based domain modelling. Integration-level domain models should be more systematically used and evaluated in programming learning technology research.

A next important question is how to construct and scale up integration-level domain models in programming learning technologies. For example, for an integration problem involving three component skills, shall it be considered as involving two paired-based integrative skills or one triple-based integrative skill? One can purely base the decisions on common teaching practices. However, to avoid expert blind spots (Nathan & Petrosino, 2003), we recommend adapting our empirical approach used in RQ2 to systematically vary problems and contrasting the performance between (higher-order) integration problems and decomposed counterparts. Alternatively, one could conduct predictive modelling on datasets with high, progressive variations in integration problems comparing different domain models (Huang et al., 2017). More importantly, analysing common errors committed on (higher-order) integration problems but not decomposed counterparts (Huang, 2018) can reveal whether a new skill arises from the integration or not to further justify the creation of a new a new higher-order integrative skill. Research into integration-level domain models may advance theories of programming language knowledge and theories of instructions for programming skills (Nelson, 2021; Xie et al., 2019).

6.3 | Integration-focused instructional design

In the current work (RQ3), our integration-focused instructional design has proven to be superior to component-focused instructional design which resembles the designs used in a large number of learning technologies. Combining the results from RQ2 and RQ3, our work

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

provides further support for the integrative skill hypothesis (Figure 5). Our preliminary mediation analysis showed that an increased amount of practice on integration problems and focused integration problems largely explained the advantageous effect of the integration-focused instructional design on students' overall learning. These results support the effectiveness of two instructional design features: providing deliberate practice and focused practice on integrative skills. We discuss our work in relation to prior work and its practical implications under each feature as follows.

One effective instructional design feature supported by our work is providing deliberate practice on integrative skills. This involves providing varied integration problems targeting different integrative skills and repeated integration problems towards mastery of each integrative skill. This is enabled by the integration-level domain model in an ITS. With only around 40 min of total practice on average, the integrative T3 already yielded significant learning benefits in both near-transfer and far-transfer situations on both integration problems and basic problems, compared to the basic T3. This indicates the potential of systematic drill practice on integration problems for the mastery of a domain through the acquisition of key integrative skills as well as deepening the learning of component skills and integrative skills. Our work adds to prior work on deliberate practice (Ericsson, 2006; Schnackenberg et al., 1998) and shows how the same principle can be applied to learning integrative skills. Our work also adds to prior research on instructions that stresses systematic integration practice. For example, Frederiksen and White (1989) have shown that a curriculum where learners practiced individual components and their integration in a stepwise fashion led to superior performance and learning than a curriculum without such a feature in psychomotor skill learning. More generally, it has been established that for complex tasks that can also be easily divided into component parts, students often learn more effectively if the components are practiced temporarily in isolation, and then progressively combined (Salden et al., 2006; White & Frederiksen, 1990; Wightman & Lintern, 1985). Unfortunately, systematic, deliberate integration practice is still absent in many current learning technologies. Our research calls for attention to and demonstrates a promising way to do so.

Another effective instructional design feature supported by our work is providing focused practice on integrative skills. Based on our prior error analysis (Huang, 2018) indicating the existence of additional integrative skills beyond component skills and cognitive load demands in integration problems, we designed focused integration problems and the integrative T3 provided a significantly higher number of such problems (Table 3). A focused integration problem (Figure 3) only requires the steps involving specific integrative skills and provides hints that explain how component skills work together, without requiring other steps that may pose cognitive load demands. Such focused problems direct students' attention to the key properties of skill integration, aligning with the instructional design guidelines from the cognitive load theory (Sweller et al., 1998). Our work corroborates the effectiveness of a prior instructional design method called Focused Practice Task Design (Huang et al., 2021; Koedinger & McLaughlin, 2010) in a new task domain. One recent study from Žanko et al. (2022) also showed positive results of focused instructions: they adapted programming lectures to spend more time on specific program examples targeting identified misconceptions and conceptual concepts, resulting in a decrease in students' mistakes related to misconceptions. Although they did not study integrative skills explicitly, their pedagogical approach is in line with our recommendation and can benefit the teaching of integrative skills.

6.4 | Limitations and future work

The current study has several limitations and elicits opportunities for future research. One unexpected finding is that we did not observe significant differences in performance accuracy on post-assessment problems between the two versions of T3. We think that on basic problems, students' scores in both conditions were likely prone to a ceiling effect (pre-assessment *Ms* ≥0.75 and post-assessment *Ms* ≥0.89), which may have left little room for improvement. Meanwhile, the integration assessment problems may be too difficult for the majority of students in both conditions (post-assessment *Ms* = 0.35), since they did not provide trace table support and involved more integrative or component skills in a single problem than most practice problems. This may have posed overly challenging far-transfer and high-fluency demands given the limited practice time. Moreover, the ITS technology and basic design shared in both conditions were already effective (RQ1), resulting in a high bar control condition. To better compare the two conditions, a future study should consider providing trace tables in assessment, providing assessment problems with a similar number of skills, and extending the study span.

Second, our pre-assessment can be further improved. The current pre-assessment problems only consisted of six basic problems due to the study time constraint, but we have enhanced the assessment of students' initial levels by utilizing the learning data. Thus, the combined data has increased the robustness of the measures of students' levels. Moving forward, we may extend the study span so that we can include more assessment problems or integrate assessment and learning together.

Third, a further mediation analysis can shed more insights into design features that promote domain mastery. For example, we may examine other problem sequencing features based on estimations of the moment of mastery (Huang et al., 2021) such as how often a tutor still selected a basic problem after a student had mastered component skills. Despite the estimation challenges (Huang et al., 2015), we may investigate whether the difference in hints for integrative skills contributes to the overall differences in outcomes (Goldin et al., 2012). We may also examine other outcome variables not examined yet.

Fourth, it is valuable to investigate the transfer of learning between integrative skills and between focused and full integration problems and think about instructional designs for promoting greater learning transfer, such as providing hints targeting the specific misconception a student has about the skill integration of a set documented

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

in our prior work (Huang, 2018), or requesting students' self-explanation on how skills integrate. This investigation may also inform whether we need to create more focused integration problems to better prepare students for full or new integration problems.

Lastly, our study is still limited in the scope and the population (e.g., we tested on beginning engineering students with 85% being male), and thus it is important to see whether results replicate in other programming topics and in other student populations, and whether socio-demographic factors moderate the intervention effects. Moreover, the investigation of skill integration can be connected to a broader range of research, such as the knowledge integration perspective based on investigations of science learning and instruction (Linn, 2005), for building theories of the learning and instruction of skill integration.

PEER REVIEW

The peer review history for this article is available at https://publons.com/publon/10.1111/jcal.12757.

DATA AVAILABILITY STATEMENT

The anonymous data is accessible through DataShop with the link upon request and submission of their intended usage: https://pslcdatashop.web.cmu.edu/DatasetInfo?datasetId=2324. All co-authors had complete access to data supporting the manuscript.

ORCID

Yun Huang https://orcid.org/0000-0002-9993-7332
Christian Schunn https://orcid.org/0000-0003-3589-297X

REFERENCES

Akram, B., Azizolsoltani, H., Min, W., Wiebe, E. N., Navied, A., Mott, B. W., Boyer, K. E., & Lester, J. C. (2020). A data-driven approach to automatically assessing concept-level CS competencies based on student programs. In *Proceedings of the CSEDM Workshop at the 13th International Conference on Educational Data Mining*. CEUR Workshop Proceedings (CEUR-WS.org).

Aleven, V., McLaren, B. M., Sewall, J., van Velsen, M., Popescu, O., Demi, S., Ringenberg, M., & Koedinger, K. R. (2016). Example-tracing tutors: Intelligent tutor development for non-programmers. *International Journal of Artificial Intelligence in Education*, *26*(1), 224–269.

Aleven, V., McLaughlin, E. A., Glenn, R. A., & Koedinger, K. R. (2016). Instruction based on adaptive learning technologies. In R. E. Mayer & P. Alexander (Eds.), *Handbook of research on learning and instruction* (pp. 522–560). Routledge.

Alibali, M. W., Stephens, A. C., Brown, A. N., Kao, Y. S., & Nathan, M. J. (2014). Middle school students' conceptual understanding of equations: Evidence from writing story problems. *International Journal of Educational Psychology*, *3*(3), 235–264.

Ambrose, S. A., Bridges, M. W., DiPietro, M., Lovett, M. C., & Norman, M. K. (2010). *How learning works: Seven research-based principles for smart teaching*. John Wiley & Sons.

Anderson, J. R., Conrad, F. G., & Corbett, A. T. (1989). Skill acquisition and the LISP tutor. *Cognitive Science*, *13*(4), 467–505.

Berges, M., & Hubwieser, P. (2015). Evaluation of source code with item response theory. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 51–56). ACM.

Brusilovsky, P. (1992). Intelligent tutor, environment and manual for introductory programming. *Educational and Training Technology International*, *29*(1), 26–34.

Čisar, S. M., Pinter, R., & Radosav, D. (2011). Effectiveness of program visualization in learning java: A case study with Jeliot 3. *International Journal of Computers Communications & Control*, *6*(4), 668–680.

Clancy, M. J., & Linn, M. C. (1999). Patterns and pedagogy. In *Proceedings of the Thirtieth SIGCSE Technical Symposium on Computer Science Education* (pp. 37–42). ACM.

Corbett, A. T., & Anderson, J. R. (1994). Knowledge tracing: Modeling the acquisition of procedural knowledge. *User Modeling and User-Adapted Interaction*, *4*(4), 253–278.

De Boeck, P., & Wilson, M. (Eds.). (2004). *Explanatory item response models: A generalized linear and nonlinear approach* (Vol. 10, pp. 978–971). Springer.

de Raadt, M. (2008). *Teaching programming strategies explicitly to novice programmers*. Doctoral dissertation, University of Southern Queensland.

Denny, P., Luxton-Reilly, A., & Simon, B. (2008). Evaluating a new exam question: Parsons problems. In *Proceedings of the Fourth International Workshop on Computing Education Research* (pp. 113–124). ACM.

du Boulay, B. (1986). Some difficulties of learning to program. *Journal of Educational Computing Research*, *2*(1), 57–73.

Ericsson, K. A. (2006). The influence of experience and deliberate practice on the development of superior expert performance. In *The Cambridge handbook of expertise and expert performance* (Vol. 38, pp. 685–705) 2–2. Cambridge University Press.

Frederiksen, J. R., & White, B. Y. (1989). An approach to training based upon principled task decomposition. *Acta Psychologica*, *71*(1–3), 89–146.

Goldberg, M. H. (2019). How often does random assignment fail? Estimates and recommendations. *Journal of Environmental Psychology*, *66*, 101,351.

Goldin, I. M., Koedinger, K. R., & Aleven, V. (2012). Learner differences in hint processing. In *Proceedings of the 5th International Conference on Educational Data Mining* (pp. 73–80). International Educational Data Mining Society.

González-Brenes, J. P., Huang, Y., & Brusilovsky, P. (2014). General features in knowledge tracing: Applications to multiple subskills, temporal item response theory, and expert knowledge. In *Proceedings of the 7th International Conference on Educational Data Mining* (pp. 84–91). International Educational Data Mining Society.

Guzdial, M. (2015). Learner-centered design of computing education: Research on computing for everyone. *Synthesis Lectures on Human-Centered Informatics*, *8*(6), 1–165.

Heffernan, N. T., & Koedinger, K. R. (1997). The composition effect in symbolizing: The role of symbol production vs. text comprehension. In *Proc. 19th Annual Conf. Cognitive Science Society* (pp. 307–312). Lawrence Erlbaum Associates.

Hertz, M., & Jump, M. (2013). Trace-based teaching in early programming courses. In *Proceedings of the 44th ACM Technical Symposium on Computer Science Education* (pp. 561–566). ACM.

Hooshyar, D., Ahmad, R. B., Yousefi, M., Yusop, F. D., & Horng, S. J. (2015). A flowchart-based intelligent tutoring system for improving problem-solving skills of novice programmers. *Journal of Computer Assisted Learning*, *31*(4), 345–361.

Höppner, F. (2019). Measuring instruction comprehension by mining memory traces for early formative feedback in Java courses. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 105–111). ACM.

Horstmann, C. S. (2020). *Big Java: Early objects* (7th ed.). John Wiley & Sons.

Hosseini, R., & Brusilovsky, P. (2013). Javaparser: A fine-grain concept indexing tool for java problems. In *CEUR Workshop Proceedings* (Vol. 1009, pp. 60–63). CEUR-WS.org.

Hsiao, I. H., Sosnovsky, S., & Brusilovsky, P. (2010). Guiding students to the right questions: Adaptive navigation support in an E-learning

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

system for Java programming. *Journal of Computer Assisted Learning*, *26*(4), 270–283.

Huang, Y. (2018). *Learner modeling for integration skills in programming* (Doctoral dissertation). University of Pittsburgh.

Huang, Y., González-Brenes, J. P., & Brusilovsky, P. (2015). Challenges of using observational data to determine the importance of example usage. In *International Conference on Artificial Intelligence in Education* (pp. 633–637). Springer.

Huang, Y., Guerra-Hollstein, J., Barria-Pineda, J., & Brusilovsky, P. (2017). Learner modeling for integration skills. In *Proceedings of the 25th Conference on User Modeling, Adaptation and Personalization* (pp. 85–93). Springer-Verlag Berlin Heidelberg.

Huang, Y., Lobczowski, N. G., Richey, J. E., McLaughlin, E. A., Asher, M. W., Harackiewicz, J. M., Aleven, V., & Koedinger, K. R. (2021). A general multi-method approach to data-driven redesign of tutoring systems. In *Proceedings of the 11th International Conference on Learning Analytics and Knowledge* (pp. 161–172). ACM.

Ihantola, P., Ahoniemi, T., Karavirta, V., & Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research* (pp. 86–93). ACM.

Jennings, J., & Muldner, K. (2020). When does scaffolding provide too much assistance? A code-tracing tutor investigation. *International Journal of Artificial Intelligence in Education*, *31*(4), 784–819.

Jennings, J., & Muldner, K. (2021). Investigating students' reasoning in a code-tracing tutor. In *The International Conference on Artificial Intelligence in Education* (pp. 203–214). Springer.

Jin, W. (2020). Automatic grading for program tracing exercises. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (p. 1409). ACM.

Kaila, E., Rajala, T., Laakso, M. J., & Salakoski, T. (2009). Effects, experiences, and feedback from studies of a program visualization tool. *Informatics in Education*, *8*(1), 17–34.

Kakeshita, T., & Ohta, K. (2019). Student log analysis functions for web-based programming education support tool pgtracer. *Journal of Information Processing*, *27*, 456–468.

Koedinger, K., & McLaughlin, E. (2010). Seeing language learning inside the math: Cognitive analysis yields transfer. In *Proceedings of the Annual Meeting of the Cognitive Science Society* (Vol. 32(No. 32)). Erlbaum.

Koedinger, K. R., & Anderson, J. R. (1993). Reifying implicit planning in geometry: Guidelines for model-based intelligent tutoring system design. In S. Lajoie & S. Derry (Eds.), *Computers as cognitive tools* (pp. 15–46). Erlbaum.

Koedinger, K. R., Anderson, J. R., Hadley, W. H., & Mark, M. A. (1997). Intelligent tutoring goes to school in the big city. *International Journal of Artificial Intelligence in Education*, *8*(1), 30–43.

Koedinger, K. R., Baker, R. S., Cunningham, K., Skogsholm, A., Leber, B., & Stamper, J. (2010). A data repository for the EDM community: The PSLC DataShop. In *Handbook of educational data mining* (Vol. 43, pp. 43–56). Routledge.

Koedinger, K. R., Corbett, A. T., & Perfetti, C. (2012). The knowledge-learning-instruction framework: Bridging the science-practice chasm to enhance robust student learning. *Cognitive Science*, *36*(5), 757–798.

Koedinger, K. R., Kim, J., Jia, J. Z., McLaughlin, E. A., & Bier, N. L. (2015). Learning is not a spectator sport: Doing is better than watching for learning from a MOOC. In *Proceedings of the Second ACM Conference on Learning@Scale* (pp. 111–120). ACM.

Koedinger, K. R., McLaughlin, E. A., & Stamper, J. C. (2012). Automated student model improvement. In *Proceedings of the 5th International Conference on Educational Data Mining* (pp. 17–24). International Educational Data Mining Society.

Koedinger, K. R., Stamper, J. C., McLaughlin, E. A., & Nixon, T. (2013). Using data-driven discovery of better student models to improve student learning. In *International Conference on Artificial Intelligence in Education* (pp. 421–430). Springer.

Kovalenko, V., Bogomolov, E., Bryksin, T., & Bacchelli, A. (2019, May). Pathminer: A library for mining of path-based representations of code. In *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)* (pp. 13–17). IEEE.

Kumar, A. N. (2015). Solving code-tracing problems and its effect on code-writing skills pertaining to program semantics. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education* (pp. 314–319). ACM.

Kumar, A. N. (2020). Long term retention of programming concepts learned using a software tutor. In *International Conference on Intelligent Tutoring Systems* (pp. 382–387). Springer.

Linn, M. (2005). The knowledge integration perspective on learning and instruction. In R. Sawyer (Ed.), *The Cambridge handbook of the learning sciences* (pp. 243–264). Cambridge University Press. https://doi.org/10.1017/CBO9780511816833.016

Lister, R., Adams, E. S., Fitzgerald, S., Fone, W., Hamer, J., Lindholm, M., McCartney, R., Moström, J. E., Sanders, K., Seppälä, O., Simon, B., & Thomas, L. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bulletin*, *36*(4), 119–150.

Liu, R., & Koedinger, K. R. (2017). Closing the loop: Automated data-driven cognitive model discoveries lead to improved instruction and learning gains. *Journal of Educational Data Mining*, *9*(1), 25–41.

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In *Proceedings of the Fourth International Workshop on Computing Education Research* (pp. 101–112). ACM.

Lovett, M. C. (2001). A collaborative convergence on studying reasoning processes: A case study in statistics. In S. Carver & D. Klahr (Eds.), *Cognition and instruction: Twenty-five years of progress* (pp. 347–384). Erlbaum.

Martin, B., Mitrovic, A., Koedinger, K. R., & Mathan, S. (2011). Evaluating and improving adaptive educational systems with learning curves. *User Modeling and User-Adapted Interaction*, *21*(3), 249–283.

Muller, O., Ginat, D., & Haberman, B. (2007). Pattern-oriented instruction and its influence on problem decomposition and solution construction. In *Proceedings of the 12th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education* (pp. 151–155). ACM.

Naps, T., Rößling, G., Brusilovsky, P., English, J., Jarc, D., Karavirta, V., Leska, C., McNally, M., Moreno, A., Ross, R. J., & Urquiza-Fuentes, J. (2005). Development of xml-based tools to support user interaction with algorithm visualization. *ACM SIGCSE Bulletin*, *37*(4), 123–138.

Nathan, M. J., & Petrosino, A. (2003). Expert blind spot among preservice teachers. *American Educational Research Journal*, *40*(4), 905–928.

Nelson, G. L. (2021). *Teaching and assessing programming language tracing* (Doctoral dissertation, University of Washington).

Neves, D. M., & Anderson, J. R. (1981). Knowledge compilation: Mechanisms for the automatization of cognitive skills. *Cognitive skills and their acquisition*.

Qi, R., & Fossati, D. (2020). Unlimited trace tutor: Learning code tracing with automatically generated programs. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education* (pp. 427–433). ACM.

Qian, Y., & Lehman, J. (2017). Students' misconceptions and other difficulties in introductory programming: A literature review. *ACM Transactions on Computing Education (TOCE)*, *18*(1), 1–24.

Rajala, T., Laakso, M. J., Kaila, E., & Salakoski, T. (2008). Effectiveness of program visualization: A case study with the ViLLE tool. *Journal of Information Technology Education*, *7*, 15.

Risha, Z., Barria-Pineda, J., Akhuseyinoglu, K., & Brusilovsky, P. (2021). Stepwise help and scaffolding for Java code tracing problems with an

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

interactive trace table. In *21st Koli Calling International Conference on Computing Education Research* (pp. 1–10). ACM.

Risha, Z., & Brusilovsky, P. (2020). Making it smart: Converting static code into an interactive trace table. In *Proceedings of Sixth SPLICE Workshop*. Virtual Event. https://cssplice.github.io/LAS20/proceedings.html

Rist, R. S. (1989). Schema creation in programming. *Cognitive Science*, *13*(3), 389–414.

Ritter, S., Anderson, J. R., Koedinger, K. R., & Corbett, A. (2007). Cognitive tutor: Applied research in mathematics education. *Psychonomic Bulletin & Review*, *14*(2), 249–255.

Rivers, K., Harpstead, E., & Koedinger, K. R. (2016). Learning curve analysis for programming: Which concepts do students struggle with? In *ICER* (Vol. 16, pp. 143–151). ACM.

Rosseel, Y. (2012). lavaan: An R package for structural equation modeling. *Journal of Statistical Software*, *48*, 1–36.

Salden, R. J. C. M., Paas, F., & van Merrienboer, J. J. G. (2006). A comparison of approaches to learning task selection in the training of complex cognitive skills. *Computers in Human Behavior*, *22*, 321–333.

Schnackenberg, H. L., Sullivan, H. J., Leader, L. F., & Jones, E. E. (1998). Learner preferences and achievement under differing amounts of learner practice. *Educational Technology Research and Development*, *46*(2), 5–16.

Shah, A., Liu, J., Stephens-Martinez, K., & Rodger, S. H. (2021). The CS1 Reviewer App: Choose your own adventure or choose for me! In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1* (pp. 331–337). ACM.

Soloway, E., & Ehrlich, K. (1984). Empirical studies of programming knowledge. *IEEE Transactions on Software Engineering*, *SE-10*(5), 595–609.

Sorva, J. (2012). *Visual program simulation in introductory programming education*. Doctoral dissertation, Aalto University.

Sorva, J., Karavirta, V., & Malmi, L. (2013). A review of generic program visualization systems for introductory programming education. *ACM Transactions on Computing Education (TOCE)*, *13*(4), 1–64.

Spohrer, J. C. (1992). *MARCEL: Simulating the novice programmer*. Intellect Books.

Spohrer, J. C., & Soloway, E. (1985). Putting it all together is hard for novice programmers. In *Proceedings of the IEEE International Conference on Systems, Man, and Cybernetics* (pp. 12–15). IEEE.

Stegeman, M. (2019). A set of exercises and tests for teaching tracing skills using a mastery approach. In *Proceedings of the 19th Koli Calling International Conference on Computing Education Research* (pp. 1–2). ACM.

Stephens-Martinez, K., Ju, A., Parashar, K., Ongowarsito, R., Jain, N., Venkat, S., & Fox, A. (2017). Taking advantage of scale by analyzing frequent constructed-response, code tracing wrong answers. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (pp. 56–64). ACM.

Sweller, J., Van Merrienboer, J. J., & Paas, F. G. (1998). Cognitive architecture and instructional design. *Educational Psychology Review*, *10*(3), 251–296.

Thomas, A., Stopera, T., Frank-Bolton, P., & Simha, R. (2019). Stochastic tree-based generation of program-tracing practice questions. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education* (pp. 91–97). ACM.

Vainio, V., & Sajaniemi, J. (2007). Factors in novice programmers' poor tracing skills. *ACM SIGCSE Bulletin*, *39*(3), 236–240.

VanLehn, K., Lynch, C., Schulze, K., Shapiro, J. A., Shelby, R., Taylor, L., Treacy, D., Weinstein, A., & Wintersgill, M. (2005). The Andes physics tutoring system: Lessons learned. *International Journal of Artificial Intelligence in Education*, *15*(3), 147–204.

Vesin, B., Ivanović, M., Klašnja-Milićević, A., & Budimac, Z. (2012). Protus 2.0: Ontology-based semantic recommendation in programming tutoring system. *Expert Systems with Applications*, *39*(15), 12229–12246.

Wang, S., Han, Y., Wu, W., & Hu, Z. (2017). Modeling student learning outcomes in studying programming language courses. In *2017 Seventh International Conference on Information Science and Technology (ICIST)* (pp. 263–270). IEEE.

Weber, G. (1996). Episodic learner modeling. *Cognitive Science*, *20*(2), 195–236.

Weber, G., & Brusilovsky, P. (2001). ELM-ART: An adaptive versatile system for web-based instruction. *International Journal of Artificial Intelligence in Education (IJAIED)*, *12*, 351–384.

White, B. Y., & Frederiksen, J. R. (1990). Causal models progressions as a foundation for intelligent learning environments. *Artificial Intelligence*, *42*, 99–157.

Wightman, D. C., & Lintern, G. (1985). Part-task training for tracking and manual control. *Human Factors*, *27*(3), 267–283.

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., Tan, A. H., Hwa, L., Li, M., & Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, *29*(2–3), 205–253.

Žanko, Ž., Mladenović, M., & Krpan, D. (2022). Analysis of school students' misconceptions about basic programming concepts. *Journal of Computer Assisted Learning*, *38*(3), 719–730.

How to cite this article: Huang, Y., Brusilovsky, P., Guerra, J., Koedinger, K., & Schunn, C. (2023). Supporting skill integration in an intelligent tutoring system for code tracing. *Journal of Computer Assisted Learning*, *39*(2), 477–500. https://doi.org/10.1111/jcal.12757

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

APPENDIX A: SKILLS AND PROBLEMS IN T3

TABLE A1 Component skills and exemplar problems in T3

ID	Label	Description	Exemplar basic problem in T3
1	//	Conduct a floor division, that is, a normal division operation except that it returns the largest possible integer.	a = 5 // 2 b = 4 // 2 c = 4 // 3

2	%	Conduct the modulo operation, that is, returns the remainder or signed remainder of a division, after one number is divided by another.
		In T3, problems involving '%' also involve 'if'.
			x = 10
			if x % 3 == 2:
			print(x, end='')
			else:
			print("yes", end='')

3	+=	Conduct compound assignment without involving loops.[a]	x = 10 i = 5 x = x + i

4	if	Use an if statement (sometimes with elif) to change the values of some variables based on some conditions.	x = 19
			y = 5
			z = 12
			min = x
			if min > y:
			min = y
			if min > z:
			min = z

5	for	Print numbers with a for loop where later iterations do not require values from earlier iterations.	for i in range(1, 3): print(i, end='')
6	a[i]	Access list elements from a list that has been initialized but has not been updated.	a = [3, 2, 7, 5] i = 2 print(a[i], end='') print(a[i+1], end='')

[a]There were other kinds of compound assignment other than += in T3, such as *=. We used the label += to refer to all kinds of compound assignments for simplicity. Also, Assignment statements like x = x + i an x + =i are both called compound assignment and were both taught in lectures before students used the tutor.

TABLE A2 Integrative skills and exemplar problems in T3

ID	Label	Description	Exemplar integrat. problem in T3

1	for & for	The integration of a for loop with another for loop where one has to print a sequence of numbers resulting from a nested for loop where the outer loop iteration variable decides the number of inner loop iterations.	for i in range(2, 4):
			for j in range(i):
			print(j, end='')
			print(';', end='')

2	for & +=	The integration of a for loop and compound assignment where one has to retrieve the updated variable from a previous iteration to conduct operations and store the result in the same variable in a new iteration in a for loop.	x = 0 for i in range(5, 7): x = x + i

3	for & if	The integration of a for loop and an if statement where one has to update a variable correctly based on conditions and retrieve the updated variable in the new iteration in a for loop.
		In T3, problems involving 'for & if' also involve 'a[i]' (list).
			lista = [10, 2, 30]
			m = lista[0]
			for i in range(1, len(lista)):
			if lista[i] < m:
			m = lista[i]

4	for & a[i]	The integration of a for loop and list access where list elements are repeatedly updated and used in latter iterations in a for loop.	b = [1, 0, 0] for k in range(1, len(b)): b[k] = b[k-1] + 2
5	a[i] & =	The integration of a list and assignment where one has to write down the updated list (not the initial list) with all its elements. In T3, problems involving 'for & a[i]' also involve 'a & =', but cells for values of specific list elements are labelled with the former, while cells requiring filling in the whole list are labelled with the latter.	b = [1, 0, 0] for k in range(1, len(b)): b[k] = b[k-1] + 2

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

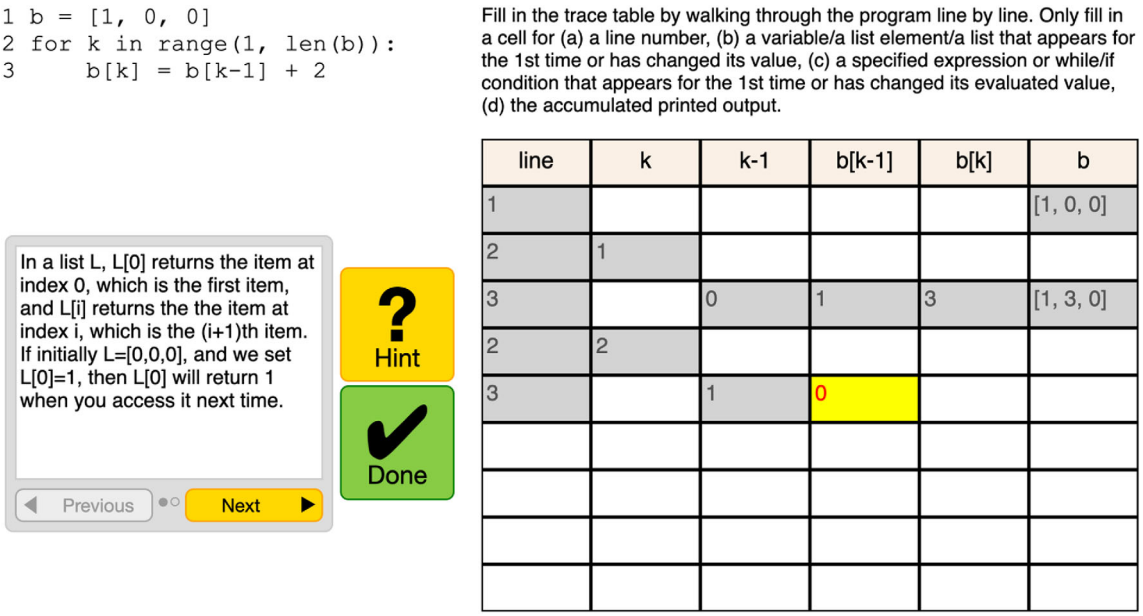


FIGURE A1 A focused integration problem in T3 requiring the integrative skill 'for & a[i]'. The displayed hint is used in the basic T3 in comparison with Figure 3

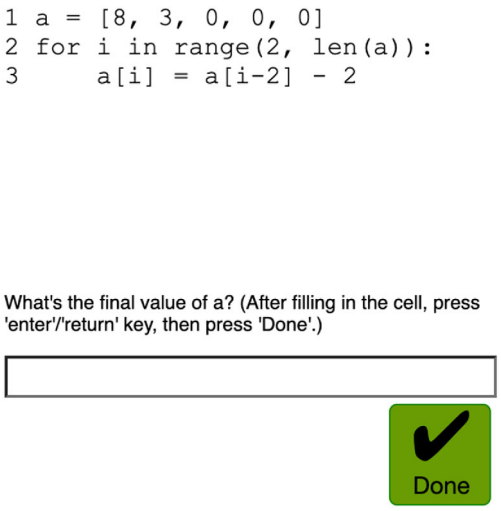


FIGURE A2 The interface for an assessment problem, the first integration problem. Each problem in the assessment displayed a small program and asked students to write down the printed output or the final value of the key variable without offering trace tables.

TABLE A3 Pre-assessment problems. All are basic problems. The numbers denote the order of the occurrence of a problem in the assessment

#1
a = 17 // 5
b = 8 // 2
print(a, end=',')
print(b, end=';')
	#2
	x = 15
	j = 5
	x = x + j
		#3
		x = 5
		y = 18
		z = 16
		result = x
		if result > y:
		result = y
		if result > z:
		result = z

#4
for k in range(4):
print(k, end='')
	#5
	a = [6, 3, 5, 7]
	i = 2
	print(a[i-1], end='')
	print(a[0], end='')
	print(a[len(a)-1], end='')
		#6
		x = 11
		if x % 5 == 2:
		print(x, end='')
		else:
		print(x % 5, end='')

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

TABLE A4 Post-assessment problems. The first six problems are basic problems and the last five problems are integration problems. The numbers denote the order of the occurrence of a problem in the assessment

#1
m = 10 // 5
n = 19 // 3
print(m, end=',')
print(n, end=';')
	#2
	x = 15
	j = 5
	x = x + j
		#3
		a = 8
		b = 10
		c = 3
		x = a
		if x < b:
		x = b
		if x < c:
		x = c

#4
for i in range(3, 5):
print(i, end='')
	#5
	a = [4, 2, 5, 7]
	i = 3
	print(a[i], end='')
	print(a[i-2], end='')
	print(a[0], end='')
	print(a[len(a)-1],
	end='')
		#6
		x = 13
		if x % 5 == 2:
		print(x, end='')
		else:
		print(x % 5, end='')

#7
a = [8, 3, 0, 0, 0]
for i in range(2,
len(a)):
a[i] = a[i-2] - 2
	#8
	a = [2, 3, 1, 1]
	for k in range(2,
	len(a)):
	for j in range(k):
	a[k] = a[k] * a[j]
		#9
		a = [11, 4, 6, 12, 20]
		z = a[0]
		y = a[0]
		for i in range(1,
		len(a)):
		if z < a[i]:
		z = a[i]
		if y > a[i]:
		y = a[i]
		x = z - y

#10
a = [1, 3]
for i in range(len(a)):
for j in range(a[i]):
print(j, end='')
print(';', end='')
	#11
	a = [1, 5, 3, 8]
	m = len(a) // 2
	k = len(a) - 1
	for i in range(m):
	t = a[i]
	a[i] = a[k-i]
	a[k-i] = t	

APPENDIX B: PROBLEM SELECTION MECHANISM IN T3

1. Update a student's learner model based on the just submitted problem.
2. Pick an unmastered skill according to following steps:
   a. Retrieve the student's latest learner model, that is, the probabilities of knowing each skill k, $P(known)_k$.
   b. Find unmastered skills, which are the skills with $P(known)_k \le m$ (the pre-specified mastery threshold); if there are no unmastered skills, stop practice.
   c. Compute the probability of an unmastered skill i being selected, $P(select)_i$, as follows, so that stronger unmastered skills are more likely to be remediated first:

$$P(select)_i = \frac{P(known)_i}{\sum_u P(known)_u},$$

   where $u \in$ unmastered skills in the skill set of the current topic.

   d. Pick a skill according to the multinomial distribution (of the skill index) specified by the set of $P(select)_i$, so that skills with higher $P(select)_i$ values have higher probabilities to be picked. Then, move to pick a problem.
3. Pick a problem focusing on remediating the chosen skill w:
   a. Compute the focus score F_{wj} for each problem j requiring the chosen skill w as follows, considering the relative strength of the skill w compared with other skills and the total amount of unlearned knowledge in the problem j:

$$F_{wj} = \frac{P(unknown)_w}{\sum_s P(unknown)_s},$$

   where $s \in$ skills of the current problem j.

   b. Pick the problem with the highest F_{wj} as the final selected problem, so that a problem with the most focus on skill w will be chosen.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License

APPENDIX C: STATISTICS OF PROBLEM TYPE TRANSITION

TABLE C1 Proportions of different types of problem transition in consecutive problem pairs of the two conditions. The mean (*SD*) over students, reduction ratio and statistical test (only for integration problem → basic problem) are reported

	Integration problem → Basic problem (pb.)					
Cond.	**Proportion**	**Difference**	**Reduction ratio**	**Basic pb. → Basic pb.**	**Basic pb. → Integ. pb.**	**Integ. pb. → Integ. pb.**
Basic T3	0.24 (0.19)	−0.08*	33%	0.35 (0.27)	0.24 (0.15)	0.17 (0.17)
Integ. T3	0.16 (0.12)			0.04 (0.08)	0.18 (0.19)	0.63 (0.25)

*$p < 0.05$.

13652729, 2023, 2, Downloaded from https://onlinelibrary.wiley.com/doi/10.1111/jcal.12757 by University Of Pittsburgh Unive, Wiley Online Library on [02/03/2023]. See the Terms and Conditions (https://onlinelibrary.wiley.com/terms-and-conditions) on Wiley Online Library for rules of use; OA articles are governed by the applicable Creative Commons License