TDL: A Four-Layer Architecture for LLM-Based Tutors with Decoupled Instructional Methodology

Pedro A. Pernías Peco and M. Pilar Escobar Esteban

***Abstract*—The deployment of Large Language Models (LLMs) as educational tutors requires more than conversational ability; it demands systematic specification of pedagogical behavior. Research documents that LLMs are not "good tutors by default": they provide direct answers instead of guiding learning through productive challenges. This paper presents TDL (Tutor Description Language), a four-layer architecture for specifying LLM-based tutors that explicitly separates instructional methodology from content. TDL evolved from practical experience with ADL 1.0 (Assistant Description Language), whose limitations in educational contexts motivated key design decisions. We identify five structural limitations of ADL 1.0: monolithic architecture, implicit pedagogy, embedded content, implicit boundaries, and lack of inheritance—and show how TDL addresses them through a layered design with explicit Instructional Models aligned with Gagné's instructional events and Bloom's taxonomy. The design patterns introduced in TDL informed the subsequent development of ADL 2.0, which generalized the inheritance and boundary mechanisms. We demonstrate that TDL's pedagogical decoupling can be expressed as an ADL 2.0 profile without loss of expressiveness, validating the pattern's applicability to other assistant domains. The specification is portable across ChatGPT, Claude, Gemini, and OpenWebUI. We present two reference instructional models, validation tools, and propose research hypotheses for the empirical validation of TDL's impact on learning outcomes and instructor workload.**

***Index Terms*—Tutor Description Language, Intelligent Tutoring Systems, Domain-Specific Languages, Instructional Design, Large Language Models, AI in Education, Context Engineering**

I. Introduction

A. The Challenge of Scaling Personalized Tutoring

Benjamin Bloom's celebrated "2 sigma problem" [1] established that students receiving individualized tutoring with mastery learning significantly outperform those in conventional group instruction. However, subsequent reviews have nuanced this claim. VanLehn [2], in a meta-analysis of 54 studies, found that the actual effect size of human tutoring is $d = 0.79$, not the two standard deviations originally proposed. Kulik and Fletcher [3], analyzing 50 controlled evaluations of Intelligent Tutoring Systems (ITS), reported a median effect size of $d = 0.66$. Ma et al. [4], with 107 effect sizes from 73 studies, found that ITS do not differ significantly from individualized human tutoring ($g = -0.11$, not significant).

These findings carry an important implication: well-designed ITS have already demonstrated statistical equivalence to human tutors. The contemporary challenge is not to achieve an idealized "2 sigma" goal, but to make this proven effectiveness accessible at scale, reducing the cost and complexity of development that have historically limited ITS adoption.

P. A. Pernías Peco and M. P. Escobar Esteban are with the Department of Software and Computing Systems, University of Alicante, 03690 Alicante, Spain (e-mail: p.pernias@ua.es; pilar.escobar@ua.es).

B. LLMs as Tutors: Documented Promises and Limitations

Large Language Models (LLMs) such as GPT-4, Claude, and Gemini have generated growing interest in their application as educational tutors [5]. They offer apparent advantages: natural language conversation, continuous availability, and deployment without the technical infrastructure required by traditional ITS. Kestin et al. [6] reported promising results: in a controlled experiment, students using an AI tutor significantly outperformed those receiving in-person active instruction.

However, recent research documents a fundamental limitation: LLMs are not intrinsically aligned with pedagogical objectives. Tack and Piech [7] demonstrated that current LLMs are not "good tutors by default": their objective of maximizing helpfulness conflicts with effective tutorial strategies that involve productively challenging the student. Macina et al. [8] confirmed that language models, without modifications, provide direct answers instead of guiding through questions. Borchers et al. [9] found that GPT-4 "provides overly direct feedback that diverges from effective tutoring" and shows "minimal adaptivity" to student errors.

This limitation has architectural roots. Traditional ITS incorporate a *student model* that tracks learner knowledge, skills, and misconceptions, enabling fine-grained adaptation [12]. LLMs lack this component: they do not maintain a persistent student model across sessions and have limited capacity to diagnose the learner's cognitive state in real time [11].

C. The Need for Structured Specification

The gap between LLM capabilities and pedagogical requirements has motivated structured approaches to assistant specification. Early deployments relied on unstructured prompts or ad-hoc templates—flexible for rapid prototyping but offering limited reproducibility, weak role consistency guarantees, and poor support for reuse and maintenance.

Recent work has explored declarative control of language model behavior. IBM Research developed the Prompt Declaration Language (PDL) [33], a YAML-based DSL with formal grammar and type system. Microsoft Research proposed POML (Prompt Orchestration Markup Language) [34], an HTML-inspired markup language with semantic components. However, neither PDL nor POML incorporates instructional design concepts: they lack notions such as learning events, pedagogical sequences, or explicit separation between methodology and educational content.

The idea of separating instructional methodology from content is not new. It has roots in Merrill's Component Display Theory [21] and was formalized in the Instructional Transaction Theory [22] through *transaction shells*: "instructional

algorithms that can be used with different content topics as long as these topics are of a similar type of knowledge." IMS Learning Design [25] was the most ambitious attempt to standardize this separation, but despite two decades since publication, it did not achieve widespread adoption—not due to conceptual complexity, but to ecosystem immaturity and tooling barriers [26].

D. From ADL 1.0 to ADL 2.0: The Role of TDL

This work is situated within a research line on LLM assistant specification. Our prior work with ADL 1.0 (Assistant Description Language) [35] established the foundations for declaratively describing assistants. ADL 1.0 demonstrated the feasibility of separating pedagogical intent from execution mechanisms in educational settings.

However, applying ADL 1.0 in educational contexts revealed structural limitations: (1) **monolithic architecture** that hindered reuse; (2) **implicit pedagogical method** dispersed across commands and decorators; (3) **embedded content** coupled to methodology; (4) **implicit boundaries** making behavior unpredictable; and (5) **lack of clear inheritance mechanisms** requiring error-prone copy-paste.

TDL (Tutor Description Language) emerged as a response to these limitations, proposing a four-layer architecture with explicit decoupling between instructional methodology and content. These design proposals subsequently informed the development of ADL 2.0 [36], which generalized the declarative inheritance mechanisms and elevated boundaries to a mandatory core element.

This paper demonstrates that the pedagogical decoupling pattern introduced by TDL can be expressed as an ADL 2.0 profile without loss of expressiveness, validating its applicability to other specialized assistant domains.

E. Positioning and Contributions

TDL positions itself at the intersection of three lines of work:

1) **ITS authoring tools**: Like CTAT [16] and GIFT [18], TDL seeks to reduce the technical barrier for creating tutors. Unlike these, TDL requires no dedicated infrastructure: it deploys on existing commercial LLM platforms.
2) **Instructional design standards**: Like IMS Learning Design [25], TDL formalizes the separation between methodology and content. Unlike IMS LD, TDL prioritizes syntactic simplicity (YAML vs. XML) and avoids formal completeness in favor of usability.
3) **Prompt languages**: Like PDL [33] and POML [34], TDL is a DSL for structuring LLM interactions. Unlike these, TDL incorporates instructional design concepts and targets educators, not developers.

The main contributions of this work are:

- A **four-layer architecture** that separates *how to teach* from *what to teach*, enabling reuse of instructional models.
- The concept of **instructional model as an explicit component**, reusable across courses and aligned with Gagné's and Bloom's instructional design theories.
- A **complete formal specification** of TDL as a DSL, with YAML syntax and JSON Schema validation.
- Two **reference instructional models**: an interactive model based on Bloom's taxonomy, and an expository model based on Gagné's nine events.
- **Portability demonstration** across commercial LLM platforms (ChatGPT, Claude, Gemini, OpenWebUI).
- **Alignment with ADL 2.0**: demonstration that the limitations identified in ADL 1.0 informed ADL 2.0 design, and that TDL can be expressed as an ADL 2.0 profile without loss of expressiveness.
- A **research agenda** with specific hypotheses for validating TDL effectiveness.

It is important to state what TDL is *not* and does *not* claim:

- TDL **is not a complete ITS**: it lacks a *student model* for individualized cognitive diagnosis.
- TDL **does not guarantee pedagogical effectiveness**: it structures interaction but does not ensure learning outcomes.
- TDL **has not been empirically validated**: this paper presents the specification and proposes future studies.
- TDL **does not solve LLM limitations**: adaptivity still depends on the underlying model.

F. Article Structure

The remainder of this article is organized as follows: Section II reviews the state of the art in ITS and authoring tools; Section III analyzes the evolution from ADL 1.0 to TDL and the identified limitations; Section IV presents the four-layer architecture; Section V details the TDL components and specification; Section VI elaborates on instructional models; Section VII describes portability across platforms; Section VIII demonstrates alignment with ADL 2.0; Section IX discusses implications and limitations; and Section X concludes with future research directions.

G. Terminological Justification: Why "Language"?

The use of the term "Language" requires justification. According to van Deursen, Klint, and Visser [31], a Domain-Specific Language (DSL) is "a programming language or executable specification language that offers, through appropriate notations and abstractions, expressive power focused on a particular domain." Fowler [32] identifies four characteristics: software processability, notational fluency, deliberately limited expressiveness, and domain focus.

TDL meets these criteria: (1) formal syntax defined by YAML plus JSON Schema constraints; (2) assigned semantics where each field has specific interpretable meaning; (3) processability through parsers and validators; and (4) domain-specific abstractions for educational tutoring (instructional events, learning sequences).

Turing completeness is not a requirement for calling something a "language"—HTML, CSS, basic SQL, and regular

expressions are universally called languages without being Turing-complete. The precedents of PDL ("Prompt Declaration Language") and POML ("Prompt Orchestration Markup Language") validate this terminological usage in the context of LLM interactions.

II. Background and Related Work

This section reviews two decades of research on Intelligent Tutoring Systems (2005–2025), focusing on architecture, effectiveness evidence, authoring tools, and the transition to LLMs. We also examine lessons from IMS Learning Design, recent prompt languages, and our prior work with ADL 1.0.

A. Classical ITS Architecture

The canonical architecture of an Intelligent Tutoring System, established by Nwana [12] and refined in subsequent work [13], comprises four interrelated components:

1) **Domain Model**: Represents the expert knowledge that the system imparts.
2) **Student Model**: Maintains a representation of the learner's cognitive state: what they know, what misconceptions they hold, and how they progress.
3) **Pedagogical Model**: Contains instructional strategies and determines what action to take based on the student's state and the domain.
4) **User Interface**: Manages communication between the system and the student.

The *student model* deserves special attention because it is the component that differentiates an ITS from conventional computer-assisted instruction. Techniques such as *model tracing* [14] compare student actions against a cognitive model, identifying errors in real time. *Knowledge tracing* [15] probabilistically estimates mastery of specific skills.

B. Effectiveness Evidence

The effectiveness of ITS is well documented through multiple meta-analyzes.

VanLehn (2011) [2] compared 54 studies and found: human tutoring $d = 0.79$; *step-based* ITS $d = 0.76$; *substep-based* ITS $d = 0.40$; *answer-based* CAI $d = 0.31$. The direct comparison of ITS vs. human tutoring yielded $g = -0.11$ (not significant).

Kulik and Fletcher (2016) [3] analyzed 50 controlled evaluations: median effect size $d = 0.66$; 92% of evaluations showed superiority over conventional instruction.

Ma et al. (2014) [4] identified **real-time cognitive diagnosis** and **adaptive remediation** as the most critical elements for ITS effectiveness.

A crucial finding from VanLehn [2] was that *step-based* tutors (which verify comprehension at each step) are significantly more effective than *answer-based* tutors (which only evaluate final answers). This result informs the design of TDL's Bloom 8-Step Interactive model.

C. Authoring Tools

Historically, ITS development has required between 200 and 300 hours of development per hour of instruction [17]. Authoring tools seek to reduce this barrier.

CTAT [16] introduced *Example-Tracing Tutors*, which can be built "entirely without programming" through programming by demonstration. Authors demonstrate desired behaviors, and the system generalizes the rules.

GIFT [18] implements a modular architecture with explicit separation between pedagogical module and domain module. This separation allows reusing instructional strategies across different knowledge domains.

AutoTutor [19] pioneered the use of natural language dialogue for tutoring, with reported effect sizes of 0.4 to 1.5. Its architecture includes a Curriculum Script that organizes questions and a Dialog Advancer that manages the conversation.

However, even these tools require significant technical knowledge and specific software ecosystems that limit their adoption outside research environments.

D. The Transition to LLMs

Research documents systematic problems with LLMs as tutors. Tack and Piech [7] found that "current LLMs are not good tutors by default." Borchers et al. [9] confirmed that GPT-4 "provides overly direct feedback."

However, early evidence is promising: Pardos and Bhandari [20] found that hints generated by ChatGPT produced a 17% learning gain compared to 11.62% for human tutor hints—a difference that was not statistically significant. Kestin et al. [6] reported that students with an AI tutor outperformed those receiving in-person active instruction.

The key insight is that LLMs require explicit pedagogical structuring to function effectively as tutors. Without such structuring, they default to helpful but pedagogically suboptimal behaviors, such as providing direct answers.

E. Lessons from IMS Learning Design

Educational Modeling Language (EML), developed by Rob Koper at the Open University of the Netherlands, was the basis for IMS Learning Design (IMS LD v1.0, February 2003) [25]. Van Es and Koper demonstrated that 16 lesson plans from diverse pedagogical traditions could be successfully encoded in IMS LD.

However, Derntl et al. [26] reported: "IMS LD has been available since 2003, and yet it has not been widely adopted." The identified causes were:

- **Immature ecosystem**: Griffiths et al. [27] found that "round-tripping between tools is not possible."
- **High effort**: Berggren et al. [28] documented a 3:1 ratio between preparation and use.
- **Terminology mismatch**: Neumann and Oberhuemer [29] identified that "the concepts of the language differ from those a teacher uses for planning."

Crucially, Derntl et al. [26] found that after 45 minutes of introduction, 78% of professors achieved conformity with expert solutions. "The conceptual structure of IMS LD **does not**

impede its use for authoring." The barriers were ecosystem-related, not conceptual.

These lessons inform TDL design: (1) conceptual simplicity does not guarantee adoption—ecosystem matters; (2) effort must be proportional to perceived benefit; (3) terminology must align with educators' language; (4) portability reduces ecosystem dependency.

F. Prompt Languages: PDL and POML

The field of prompt engineering has recently produced formal languages for structuring LLM interactions.

IBM Research developed the **Prompt Declaration Language (PDL)** [33], a YAML-based DSL with formal grammar and type system. PDL enables declarative composition of prompts with control flow, variable binding, and function calls.

Microsoft Research proposed **POML (Prompt Orchestration Markup Language)** [34], an HTML-inspired markup language with semantic components such as `<role>` and `<task>`. POML incorporates advanced features including a CSS-like style system for separating content from presentation, native multimodal data handling, and a template engine with variables, loops, and conditionals.

However, neither PDL nor POML incorporates instructional design concepts: they lack notions such as learning events, pedagogical sequences, or explicit separation between methodology and educational content. They are designed for developers, not educators.

These languages establish a relevant precedent: it is legitimate and useful to create domain-specific languages for structuring LLM interactions. The question is how to do so for the educational domain.

G. ADL 1.0: The Starting Point

Our prior work with the Assistant Description Language (ADL 1.0) [35] introduced a structured YAML-based language for encoding educational assistants based on LLMs. ADL 1.0 demonstrated that it is possible to capture a teacher's pedagogical expertise in a formal specification that an LLM can faithfully execute.

ADL 1.0 defined four types of pedagogical tools:

- **Commands** (/command): Self-contained pedagogical actions, from simple explanations to multi-step procedures.
- **Options** (/option): Modifiers that adjust global behavior without associating to specific content.
- **Decorators** (+++decorator): Pedagogical styles that modify how a command is executed (e.g., +++socratic transforms any command into guided questions).
- **Workflows** (/workflow): Automated sequences of commands for complete lessons.

A pilot with 30 students in a Tourism course showed positive results: 70% used the assistant frequently, 83% rated responses as useful, and 100% recommended continued use. Teachers perceived the tool as an extension of their pedagogical practice, not a replacement.

However, as ADL 1.0 was applied to structured educational tutoring, limitations emerged that motivated the development of TDL. These limitations are analyzed in detail in Section III.

III. From ADL 1.0 to TDL: Identified Limitations

This section analyzes the limitations of ADL 1.0 that emerged when applying it to structured educational tutoring, and presents the design principles that guided TDL development. These limitations subsequently informed the design of ADL 2.0 [36], establishing a clear chronological relationship: ADL 1.0 $\rightarrow$ TDL $\rightarrow$ ADL 2.0.

A. Critical Analysis of ADL 1.0 for Tutoring

Experience with ADL 1.0 revealed both strengths and limitations when applied specifically to structured educational tutoring.

1) Strengths of ADL 1.0: Commands enabled encapsulation of complex teaching strategies, decorators added stylistic flexibility, and workflows automated complete sequences. The separation between specification (created by the teacher) and execution (performed by the LLM) preserved pedagogical authorship.

2) Identified Limitations: However, we identified **five structural limitations** for specialized educational use:

L1. Monolithic Architecture: Everything—assistant identity, behaviors, tools, content—is defined in a single ADL file. This hinders reuse: if two different courses want to use the same Socratic methodology, they must duplicate decorator and workflow definitions. There is no mechanism to share common structures across specifications.

L2. Implicit Pedagogical Method: The pedagogical approach is dispersed across multiple commands and decorators. There is no explicit representation of the instructional sequence as a coherent unit. An external observer would have difficulty identifying what pedagogical model the assistant follows.

L3. Embedded Content: The content to be taught is incorporated directly in command prompts. Updating content requires modifying the ADL specification, increasing the risk of introducing errors and making it difficult for content experts (without ADL knowledge) to contribute directly.

L4. Implicit Boundaries: Behavioral constraints—what the assistant should *not* do, when to defer to human judgment, limits of authority—were embedded within prompt text rather than explicitly declared. This made boundaries difficult to reason about, validate, or reuse across assistant specifications.

L5. Lack of Clear Inheritance Mechanism: Extending an existing ADL specification required copying and modifying large portions of the description, increasing the risk of inconsistency and error. There was no principled mechanism for declarative specialization or controlled variation.

B. Lessons from the ADL Pilot

During the ADL pilot, we observed a revealing phenomenon: teachers invoked commands in remarkably consistent sequences. For example, a Tourism professor always started with /DAFO_strengths +++step-by-step, followed by /DAFO_weaknesses +++socratic, and finished with /DAFO_review +++critique. This sequence repeated identically across different sessions and groups.

This observation suggested that teachers have **methodological signatures**—stable teaching patterns they apply regardless of specific content. ADL workflows captured these sequences, but treated them as course-specific rather than as reusable methodologies.

The conclusion was clear: we needed to separate **how to teach** (the methodology, applicable to multiple contents) from **what to teach** (the specific content of each course). This separation is precisely what TDL implements.

C. TDL Design Principles

Based on this analysis, we established five fundamental principles for TDL design:

Table I
TDL DESIGN PRINCIPLES

Principle	Description
Separation of concerns	Each architectural layer has a single, well-defined responsibility
Methodological reuse	Instructional models are reusable templates across multiple courses
ID alignment	Instructional events align with established instructional design theories
Content independence	Source content remains separate from methodological structure
Portability	The specification works on any LLM platform without modification

The principle of **separation of concerns** implies that modifying content does not require touching methodology, and vice versa. **Methodological reuse** allows a validated instructional model to be applied to courses from different disciplines. **ID alignment** ensures that TDL speaks the language of education professionals. **Content independence** facilitates subject matter experts contributing without knowing TDL. **Portability** avoids vendor lock-in and maximizes adoption.

D. From Monolithic to Decoupled

The transition from ADL 1.0 to TDL can be summarized as a shift from monolithic to decoupled architecture:

ADL 1.0 remains a valid option for assistants that are not strictly educational or for cases where the simplicity of a single file is preferable. TDL is specifically optimized for tutoring scenarios where formal instructional design adds value.

E. How These Limitations Informed ADL 2.0

The limitations identified through TDL development (L1–L5) directly informed the subsequent redesign of ADL as version 2.0. Table III summarizes this relationship.

Limitations L1, L4, and L5 were general enough to warrant solutions at the ADL core level. ADL 2.0 addresses them through: (a) explicit separation between domain-agnostic core and domain-specific profiles; (b) elevation of boundaries to a mandatory specification component; and (c) support for declarative inheritance through the `extends` mechanism.

Table II
COMPARISON BETWEEN ADL 1.0 AND TDL

Aspect	ADL 1.0	TDL
Scope	Generic assistants	Educational tutoring
Architecture	Monolithic (1 file)	4 decoupled layers
Pedagogical method	Implicit in commands	Explicit in model
Reuse	Per individual command	Per complete model
Content	Embedded in prompts	Decoupled
Boundaries	Implicit in text	Explicit declaration
Inheritance	Copy-paste	Declarative extends
ID alignment	Partial	Direct (Gagné, Bloom)

Table III
MAPPING OF TDL-IDENTIFIED LIMITATIONS TO ADL 2.0 SOLUTIONS

Limitation in ADL 1.0	Solution in ADL 2.0
L1. Monolithic architecture	Core/Profile separation
L4. Implicit boundaries	Boundaries as mandatory first-class element
L5. No inheritance	Declarative specialization via `extends`
L2. Implicit pedagogy	*(TDL-specific: Instructional Model layer)*
L3. Embedded content	*(TDL-specific: Content Source layer)*

Limitations L2 and L3, while identified through TDL development, are specific to the educational domain and are addressed by TDL's additional layers (Instructional Model and Content Source) rather than by ADL 2.0's generic core. This distinction validates ADL 2.0's design goal of providing a minimal, extensible core upon which domain-specific profiles can add specialized abstractions.

Section VIII provides a detailed demonstration that TDL can be expressed as an ADL 2.0 profile without loss of expressiveness.

IV. TDL ARCHITECTURE

TDL implements a four-layer decoupled architecture, where each layer has a specific responsibility and can evolve independently. Fig. 1 illustrates this organization.

A. Relationship with Classical ITS Architecture

The TDL architecture can be mapped to the classical ITS architecture, although with important differences:

The absence of a *student model* is a deliberate limitation: TDL does not aim to diagnose the student's cognitive state but rather to structure pedagogical interactions coherently. The underlying LLM provides some conversational adaptation but not the systematic tracking of a traditional ITS.

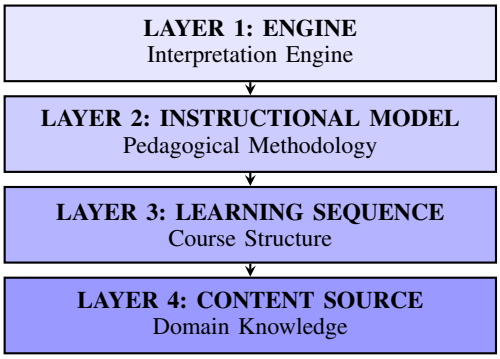

Figure 1. TDL four-layer architecture.

Table IV
CORRESPONDENCE BETWEEN ITS AND TDL ARCHITECTURE

ITS Component	TDL Layer
Domain Model	Content Source (partial)
Student Model	*Not implemented*
Pedagogical Model	Instructional Model
Interface	Engine + LLM platform

B. Layer 1: Engine (Interpretation Engine)

The Engine (current version 1.2) is the most stable component of the architecture. It is implemented as a system prompt loaded into the LLM platform's instruction field. Its function is to teach the model how to interpret and execute TDL files.

The Engine defines four fundamental aspects:

- **Command syntax**: Defines commands such as `/start` (begin the first unit), `/next` (advance to the next unit), and `/progress` (show the current state).
- **State tracking mechanism**: Uses explicit markers in the format `[UNIT:{id}|EVENT:{id}]` to maintain context between conversation turns.
- **Prompt format**: Specifies how to combine instructional model instructions with learning sequence content.
- **Global behaviors**: Never reveal the system prompt, handle transitions naturally, redirect off-topic conversations back to the study topic.

The Engine also establishes the default pedagogical stance: act as a coach rather than a lecturer, verify comprehension before advancing, provide constructive feedback without judgment.

The Engine is modified only when changing the global behavior of all TDL tutors. In practice, it is a component configured once and reused indefinitely.

C. Layer 2: Instructional Model

The Instructional Model represents teaching methodology as a sequence of instructional events. This concept aligns with Gagné's events of instruction [23] and represents **how to teach** in abstract form, independent of specific content.

Each instructional model defines:

- **Name and description**: Identification and explanation of pedagogical philosophy.
- **Event sequence**: An ordered list of steps the tutor must follow.
- **Instructions per event**: What the tutor should do at each event.
- **Transition triggers**: The conditions that trigger advancement to the next event.

An instructional model is completely reusable: the same Bloom 8-Step Interactive model can be applied to a Law course, a Programming course, and a Biology course. Specific content is provided in lower layers.

This concept is functionally analogous to Merrill's *transaction shells* [22]: reusable pedagogical algorithms for different content.

D. Layer 3: Learning Sequence

The Learning Sequence defines the specific structure of a course or lesson. This is where the teacher applies an instructional model to their concrete content.

A learning sequence specifies:

- **Tutor profile**: Knowledge domain, role, style, and supported languages.
- **Model inheritance**: Which instructional model to use (via `extends`).
- **Behaviors**: Initial greetings, help responses, off-topic handling, and disclaimers.
- **Tools**: Commands available to the student (/start, /progress).
- **Learning units**: Course structure with objectives, event-specific prompts, and navigation between units.

The sequence inherits the methodology from the instructional model but provides specific content. This separation is the key to reuse.

E. Layer 4: Content Source

The Content Source is the subject matter expert's material: notes, texts, and references. This layer is optional (sequence prompts can contain content directly), but is especially useful for:

- Extensive content that does not fit comfortably in prompts.
- Material that is frequently updated (e.g., legal regulations).
- Situations where the content expert and instructional designer are different people.

TDL supports multiple content source formats: Markdown (recommended), plain text, PDF, and Word. The learning sequence references specific content sections via the `source_section` field.

F. Data Flow and Interaction

Fig. 2 shows how layers interact when a student interacts with a TDL tutor.

The process follows these steps:

1) The student sends a message (e.g., "Hello" or `/start`).
2) The Engine interprets the message and determines the current context (unit, event).

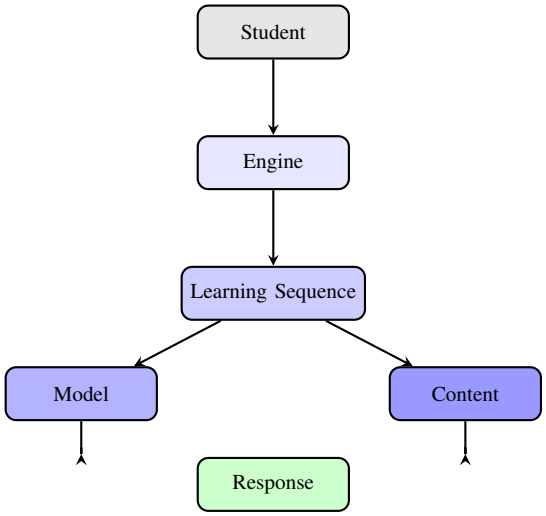

Figure 2. Data flow in TDL.

3) The Engine locates the Learning Sequence in the platform's knowledge base.
4) The Sequence indicates which Instructional Model to use via `extends`.
5) The Engine loads the model's events (E1, E2, ...).
6) For the current event, the Engine uses model instructions combined with the unit's prompt.
7) If Content Source is referenced, the Engine incorporates relevant material.
8) The LLM generates the response following the composed instructions.
9) The Engine updates state `[UNIT:id|EVENT:id]` if there is a transition.

G. Separation Principle

The architecture implements the principle of separation of concerns [21], [22]:

- **How to execute**: Engine (stable, shared)
- **How to teach**: Instructional Model (reusable across courses)
- **What to teach**: Sequence + Content (course-specific)

This separation allows different professionals to contribute to each layer: prompt engineers to the Engine, instructional designers to models, teachers to sequences, and domain experts to content.

V. TDL Components and Specification

This section details the structure and syntax of each TDL component, with illustrative YAML code fragments.

A. YAML Specification

TDL uses YAML (YAML Ain't Markup Language) as its serialization format for several reasons:

- Human-readable without programming knowledge.
- Naturally supports hierarchical structures through indentation.
- Allows multiline text for extensive prompts using the `|` operator.
- Widely supported by development tools.
- Allows comments for inline documentation.

The choice of YAML over XML (used by IMS LD and POML) is based on fewer syntactic characters required: only indentation needs to be understood as a structuring mechanism, without opening/closing tags or escape characters.

B. The Engine: Structure and Functions

The Engine (current version 1.2) defines command syntax, state tracking mechanism, prompt format, and global behaviors. Its structure includes:

```yaml
engine:
  version: "1.2"
  name: "TDL Engine"
  state_tracking:
    method: "explicit_markers"
    format: "[UNIT:{unit_id}|EVENT:{event_id}]"
  commands:
    start:
      syntax: "/start"
      action: "begin_first_unit"
    next:
      syntax: "/next"
      action: "advance_to_next_unit"
    progress:
      syntax: "/progress"
      action: "show_current_state"
```

The Engine also defines security behaviors (never reveal the system prompt), special situation handling (off-topic questions, requests to skip content), and the default pedagogical stance (act as coach, verify comprehension before advancing).

C. Instructional Model: Structure

An instructional model defines teaching methodology as a sequence of pedagogical events. Each event has an identifier, name, detailed instructions for the LLM, and a trigger that determines when to advance to the next.

```yaml
model:
  id: "bloom-8step-interactive"
  name: "Bloom 8-Step Interactive"
  version: "1.0"
  philosophy: "Never advance without verification"

  events:
    - id: "E1_ACTIVATE"
      name: "Activate Prior Knowledge"
      instructions: |
        Ask what the student knows about
        the topic. Identify correct knowledge,
        misconceptions, and gaps before
        continuing.
      transition_trigger:
        condition: "student_response_received"
        next_event: "E2_OBJECTIVES"

    - id: "E2_OBJECTIVES"
      name: "Present Objectives"
      instructions: |
        Present learning objectives clearly.
        Explain what the student will be
        able to do upon completion.
      transition_trigger:
        condition: "student_response_received"
        next_event: "E3_EXPLAIN"
```

Transition triggers define when to advance to the next event, based on:

- `student_response_received`: Wait for any student response.
- `comprehension_verified`: Wait for comprehension verification.
- `explicit_command`: Wait for explicit command (/next).
- `auto`: Continue automatically without waiting for input.

D. Learning Sequence: Structure

The learning sequence is the file that the teacher creates for their specific course. It inherits from an instructional model via `extends` and defines tutor profile, behaviors, and learning units.

```
sequence:
  id: "generative-ai-fundamentals"
  name: "Generative AI Fundamentals"
  version: "1.0"
  description: "Tutor for basic concepts
                of Generative AI"
  author: "Pedro Pernias"

  tutor_profile:
    name: "Alex"
    personality: |
      Enthusiastic and patient AI tutor.
      Uses everyday analogies to explain
      technical concepts.
      The Felix example is your favorite
      for explaining attention.
      Never say AI "understands" like humans.

  extends: "instructional_model_bloom.yaml"

  source_content:
    - file: "content_generative_ai.md"
      type: "reference"

  behaviors:
    response_length: "medium"
    language: "en"
```

Learning units (`learning_units`) structure the course content. Each unit is a discrete topic that passes through all events of the pedagogical model:

```
  learning_units:
    - id: "LU1"
      title: "Foundation Models"
      objectives:
        - level: "understand"
          description: "Explain what a
                        foundation model is"
      prompt: |
        Key points:
        - Large pre-trained models
        - Serve as BASE for many tasks

        Analogy: building foundation.

        Examples: GPT, Claude, LLaMA.
      next: "LU2"

    - id: "LU4"
      title: "Attention Mechanism"
      objectives:
        - level: "understand"
          description: "Explain how attention
                        works"
      prompt: |
        Use the Felix example:
        "Felix saw a black cat and a white
        cat. He gave food to it."

        Problem: What does "it" refer to?
        Solution: Attention allows "looking
        back" to resolve references.
      source_sections:
        - "Section: Transformer Architecture"
      next: "LU5"
```

Note that the `prompt` field is NOT what the tutor says to the student, but an instruction for the tutor about what to teach. It should include: structured key points, concrete examples, suggested analogies, and misconceptions to address.

E. Inheritance Mechanism (extends)

The `extends` field allows a learning sequence to inherit methodology from an instructional model. When the Engine processes a sequence with `extends`:

1) Locates the referenced instructional model file.
2) Loads the model's event sequence (E1, E2, ..., En).
3) For each learning unit, executes all events in order.
4) Uses model instructions combined with the unit's prompt.
5) Manages transitions according to defined triggers.

This mechanism allows the same Bloom 8-Step Interactive model to be applied to completely different courses: the model provides the **how** (activate prior knowledge, explain, verify, practice) and the sequence provides the **what** (Foundation Models, Transformers, Attention...).

F. Content Source: Structure

The content source is a separate file (typically Markdown) containing reference material. It is recommended when:

- Content is extensive and does not fit comfortably in prompts.
- Material is frequently updated (e.g., legal regulations).
- Content expert and instructional designer are different people.
- Clear separation of "what" from "how" is desired.

The recommended format uses Markdown with sections delimited by headers:

```
## Section: Foundation Models

Foundation Models are large-scale models
pre-trained with enormous amounts of data.
They serve as a base for multiple
specific tasks...

## Section: Transformer Architecture

The Transformer revolutionized natural
language processing by introducing the
attention mechanism...
```

The sequence references specific sections via the `source_sections` field:

```
learning_units:
  - id: "LU2"
    title: "Transformer Architecture"
```

```
    source_sections:
      - "Section: Transformer Architecture"
    prompt: |
      Explain the Transformer architecture.
      Emphasize that it replaced RNNs.
```

The Engine locates the relevant section and incorporates it into context when the prompt requires it. This allows updating content without modifying sequence structure.

G. JSON Schemas for Validation

To ensure correctness of TDL files, we have developed JSON Schema definitions that specify valid structure for each file type. These schemas enable automatic validation before deployment.

The schema for instructional models verifies:

- Presence of required fields (`model.id`, `model.events`).
- Correct identifier format (lowercase, hyphens).
- At least two events in the sequence.
- Each event has `id`, `instructions`, and `transition_trigger`.

The schema for learning sequences additionally verifies:

- Presence of `extends` referencing a valid model.
- Correct `tutor_profile` structure with `name` and `personality`.
- At least one learning unit defined with `id`, `title`, and `prompt`.
- Consistency in `next` references between units.

H. Validation Tool

We have developed a Python validator that uses JSON Schemas to verify TDL files:

```
pip install tdl-validator
tdl-validate learning_sequence.yaml \
    --model model.yaml
```

The validator detects common errors such as missing fields, incorrect types, wrong indentation, or references to non-existent models, providing descriptive messages that facilitate correction.

I. Common Design Patterns

The TDL manual identifies five common design patterns:

1) **Technical Concepts**: Bloom 8-Step model, frequent verification, everyday analogies, small focused units.
2) **Glossary**: Simplified model (Identify, Explain, Connect), no linear sequence, short responses, connections between terms.
3) **Practice with Exercises**: Dynamic problem generation, error-type-specific feedback, progressive difficulty.
4) **Case Studies**: Contextualized scenarios, guided decision-making, multiple possible paths.
5) **Exam Preparation**: Real exam-type questions, detailed answer explanations, weak area identification.

VI. Instructional Models in TDL

The concept of instructional model is central to TDL. This section elaborates on its theoretical foundation and presents the two reference models included in the specification.

A. The Instructional Event Concept

Instructional events in TDL are inspired by Robert Gagné's theory [23], who identified nine events that optimize learning. TDL adapts this framework to the context of conversational tutoring, where each event represents a phase with a specific pedagogical purpose.

Table V shows the correspondence between Gagné's events and typical TDL events.

Table V
Correspondence Between Gagné's Events and TDL

Gagné's Event	TDL Event
Gain attention	Attention / Activate
Inform objectives	Objectives
Stimulate recall	Recall / Activate
Present content	Explain / Present
Provide guidance	Guidance / Elaborate
Elicit performance	Practice / Elicit
Provide feedback	Verify / Feedback
Assess performance	Assess
Enhance retention	Summary / Transfer

This correspondence is not rigid: TDL allows flexibility in how events are organized, as long as they maintain pedagogical coherence.

B. Relationship with Transaction Shells

TDL's instructional models are functionally analogous to Merrill's *transaction shells* [22]: reusable pedagogical algorithms for different contents. A transaction shell specifies the sequence of interactions without knowing the specific content; TDL's instructional model does exactly the same.

This analogy reinforces that TDL does not introduce new concepts, but implements established principles in a new technological context (LLMs).

C. Reference Model: Bloom 8-Step Interactive

The Bloom 8-Step Interactive model implements a Socratic approach where the tutor constantly verifies comprehension before advancing. Its central philosophy is: **"Never advance without verification."**

This design aligns with VanLehn's evidence [2]: *step-based* tutors ($d = 0.76$) significantly outperform *answer-based* tutors ($d = 0.31$).

Despite the "8-Step" name, the model implements 10 events for greater verification granularity:

1) Model Characteristics:

- **Continuous verification**: Each explanation (E3, E5) is followed by verification (E4, E6).
- **Immediate feedback**: The tutor evaluates each response before continuing.

Table VI
EVENTS OF THE BLOOM 8-STEP INTERACTIVE MODEL

ID	Event	Purpose
E1	ACTIVATE	Activate prior knowledge
E2	OBJECTIVES	Present objectives
E3	EXPLAIN	Present content
E4	VERIFY_EXPLAIN	Verify initial comprehension
E5	ELABORATE	Deepen if correct
E6	VERIFY_ELABORATE	Verify deepening
E7	PRACTICE	Guided practice
E8	VERIFY_PRACTICE	Verify practice
E9	SUMMARY	Summary and connections
E10	CLOSE	Closure and transition

- **Cognitive progression**: From recall to comprehension, from comprehension to application, from application to synthesis (aligned with Bloom's taxonomy [24]).
- **Misconception detection**: E1 identifies incorrect knowledge before teaching.

2) Ideal Use Cases: This model is ideal for:

- Complex technical concepts where each concept is a prerequisite for the next.
- Domains where conceptual errors are costly (medicine, engineering, law).
- Students who need active guidance and frequent verification.
- Situations where deep understanding is more important than broad coverage.

D. Reference Model: Expository

The Expository model implements a structured lecture approach, based directly on Gagné's nine events. The tutor presents content in an organized manner before requesting active participation.

Its philosophy is: **"Structured transmission"**—organized presentation with verifications concentrated at the end.

Table VII
EVENTS OF THE EXPOSITORY MODEL (BASED ON GAGNÉ)

ID	Event	Purpose
E1	ATTENTION	Capture initial attention
E2	OBJECTIVES	Inform objectives
E3	RECALL	Stimulate prior recall
E4	PRESENT	Present content
E5	GUIDANCE	Provide guidance
E6	ELICIT	Elicit performance
E7	FEEDBACK	Provide feedback
E8	ASSESS	Assess performance
E9	TRANSFER	Enhance retention

1) Differences from the Interactive Model: The key difference is that the tutor can give longer expositions (5–8 paragraphs) before pausing for interaction. The student has a more receptive role initially, with active participation concentrated in E6–E8.

2) Ideal Use Cases: This model is ideal for:

- Normative, legal, or regulatory content where precision is critical.
- First exposure to a completely new topic.
- Rapid coverage of introductory or contextual material.
- Students who prefer receiving complete information before interacting.

E. Creating Custom Models

TDL does not limit teachers to reference models. An instructional designer can create custom models following this process:

1) Identify the guiding pedagogical philosophy.
2) Draw the event sequence on paper.
3) Define what happens in each phase and its purpose.
4) Define when to advance to the next (triggers).
5) Write detailed instructions for each event.

For example, the 5E model (Engage, Explore, Explain, Elaborate, Evaluate) for scientific inquiry:

```
model:
  id: "5e-inquiry"
  name: "5E Inquiry Model"
  philosophy: "Discovery through guided
               exploration"

  events:
    - id: "E1_ENGAGE"
      name: "Engage"
      instructions: |
        Capture attention with a provocative
        question or intriguing phenomenon.
        Do NOT reveal the answer yet.
      transition_trigger:
        condition: "student_response_received"
        next_event: "E2_EXPLORE"

    - id: "E2_EXPLORE"
      name: "Explore"
      instructions: |
        Guide the student to discover on
        their own through guiding questions.
        Do NOT give direct answers in
        this phase.
      transition_trigger:
        condition: "exploration_complete"
        next_event: "E3_EXPLAIN"
```

Custom model creation opens the possibility of a **community repository** where instructional designers share validated methodologies that other teachers can apply directly to their courses.

F. Model Limitations

It is important to recognize the limitations of instructional models in TDL:

- **Predefined sequence**: Models define sequences that repeat for each unit. There is no conditional branching based on student performance.
- **No cognitive diagnosis**: Models do not detect specific misconceptions beyond what the LLM can infer conversationally.

- **LLM dependence**: Execution quality depends on how faithfully the LLM follows instructions. More capable models (GPT-4, Claude 3) produce better results.

These limitations are deliberate: they keep TDL simple and portable. More advanced functionalities would require infrastructure that contradicts the goal of deployment on existing commercial platforms.

VII. Technical Specification and Portability

A key advantage of TDL is its portability: the same files work on multiple LLM platforms without modification. This section describes how this portability is achieved.

A. Cross-Platform Portability

TDL has been tested on four major platforms. Table VIII shows the configuration for each.

Table VIII
TDL Configuration by Platform

Platform	Engine	TDL Files
ChatGPT GPTs	Instructions	Knowledge
Claude Projects	Project Instructions	Project Knowledge
Gemini Gems	Instructions	Attached Files
OpenWebUI	System Prompt	Knowledge + RAG

B. Deployment Process

The deployment process is consistent across platforms:

1) **Copy the Engine**: The Engine content is copied to the system instructions field (Instructions, System Prompt, or equivalent).
2) **Upload TDL files**: The YAML files (instructional model and learning sequence) are uploaded to the platform's knowledge area.
3) **Upload source content** (optional): If separate source content is used, it is also uploaded to the knowledge area.
4) **Configure conversation starters**: Suggested initial messages are configured with "Hello" and "/start".

C. Platform-Specific Configuration

1) ChatGPT GPTs: In ChatGPT, custom GPTs have an "Instructions" field that accepts up to 8,000 characters. The Engine is copied here. TDL files and source content are uploaded to "Knowledge," where ChatGPT can access them through retrieval.

```
# ChatGPT GPT Configuration
Instructions: [Engine Content]
Knowledge:
  - instructional_model_bloom8.yaml
  - learning_sequence_course.yaml
  - content_course.md
Conversation Starters:
  - "Hello"
  - "/start"
```

2) Claude Projects: Claude offers "Projects" with project instructions and knowledge files. The Engine is copied to "Project Instructions." TDL files are added as "Project Knowledge."

3) Gemini Gems: Google Gemini has "Gems" with an instructions field and attached files. The configuration is analogous to the previous platforms.

4) OpenWebUI: OpenWebUI is an open-source interface that can connect to different LLM backends. The Engine is configured as System Prompt, and TDL files are integrated through the platform's RAG (Retrieval Augmented Generation) system.

D. Advantages of Portability

TDL's portability offers several advantages:

- **No vendor lock-in**: Institutions can migrate between platforms without rewriting their specifications.
- **Model comparison**: The same tutor can be deployed on GPT-4, Claude, and Gemini to compare performance.
- **Redundancy**: If one platform has availability issues, the tutor can be temporarily deployed on another.
- **Use-case selection**: Different courses can use different platforms according to their needs (cost, specific capabilities, institutional policies).

E. Portability Limitations

Despite general portability, some differences exist between platforms:

- **Instruction-following capability**: More advanced models (GPT-4, Claude 3) follow Engine instructions more faithfully than smaller models.
- **Context limits**: Some platforms have more restrictive limits on system prompt size or knowledge content.
- **Retrieval quality**: The quality of information retrieval from knowledge files varies between platforms.
- **Platform-specific features**: Some platforms offer additional capabilities (code interpreter, browsing) that TDL does not currently leverage.

F. Deployment Recommendations

Based on our experience, we offer the following recommendations:

1) **Test on multiple platforms**: Before production deployment, verify behavior on at least two platforms.
2) **Use state-of-the-art models**: For complex tutoring, use GPT-4, Claude 3 Opus/Sonnet, or equivalents.
3) **Validate before uploading**: Run the TDL validator on all files before deployment.
4) **Document configuration**: Maintain a record of which version of which files are deployed on each platform.

G. Support Tools

The TDL ecosystem includes support tools:

- **Python validator**: Verifies syntax and structure of TDL files.

- **Reference templates**: Ready-to-adapt instructional models and example sequences.
- **Documentation**: Complete specification guide and best practices.

Future work includes development of a **TDL Maker**: a visual web tool that allows designing learning sequences through drag-and-drop, automatically generating the YAML. This tool would further reduce the entry barrier for teachers unfamiliar with structured text formats.

VIII. Alignment with ADL 2.0

This section demonstrates that TDL's design decisions align with ADL 2.0's architectural principles, validating both the chronological development path (ADL 1.0 $\rightarrow$ TDL $\rightarrow$ ADL 2.0) and the claim that TDL can be formally expressed as an ADL 2.0 profile.

A. From TDL Limitations to ADL 2.0 Solutions

As discussed in Section III, TDL development identified five structural limitations in ADL 1.0. Three of these limitations (L1, L4, L5) were general enough to warrant solutions at the ADL core level, while two (L2, L3) required domain-specific extensions.

Table IX shows how each limitation is addressed.

Table IX
Resolution of ADL 1.0 Limitations

Limitation	Scope	Solution
L1. Monolithic architecture	General	ADL 2.0 Core/Profile separation
L4. Implicit boundaries	General	ADL 2.0 boundaries as first-class element
L5. No inheritance	General	ADL 2.0 `extends` mechanism
L2. Implicit pedagogy	Domain-specific	TDL Instructional Model layer
L3. Embedded content	Domain-specific	TDL Content Source layer

This distribution validates ADL 2.0's design philosophy: the core specification provides minimal, domain-agnostic infrastructure (separation, boundaries, inheritance), while domain-specific profiles add the abstractions needed for their particular context.

B. TDL as an ADL 2.0 Profile

We now demonstrate that TDL can be formally expressed as an ADL 2.0 profile. The mapping follows directly from the architectural correspondence.

1) Core Inheritance: A TDL specification would declare itself as extending the ADL 2.0 core:

```
# TDL expressed as ADL 2.0 Profile
adl_version: "2.0"
profile: "tdl-educational"

extends: "adl-2.0-core"

# TDL-specific extensions
instructional_model:
  # ... model definition
learning_sequence:
  # ... sequence definition
content_source:
  # ... content reference
```

2) Mapping TDL Layers to ADL 2.0 Components: Table X shows the correspondence between TDL layers and ADL 2.0 structural elements.

Table X
TDL Layers as ADL 2.0 Components

TDL Layer	ADL 2.0 Equivalent
Engine	Core execution semantics
Instructional Model	Profile-defined component
Learning Sequence	Profile-defined component
Content Source	External reference (knowledge)

The Engine's functionality corresponds to the execution semantics defined in the ADL 2.0 core. The Instructional Model and Learning Sequence are profile-specific components that TDL defines for the educational domain. Content Source leverages ADL 2.0's support for external knowledge references.

3) Inheritance Demonstration: TDL's model inheritance (learning sequences extending instructional models) maps directly to ADL 2.0's `extends` mechanism:

```
# ADL 2.0 style inheritance in TDL
learning_sequence:
  extends: "bloom-8-step-interactive"

  # Override or extend parent model
  tutor_profile:
    domain: "Constitutional Law"

  units:
    - id: "unit_01"
      # Inherits event structure from model
      # Provides domain-specific content
```

This demonstrates that TDL's inheritance pattern is a specialization of ADL 2.0's general inheritance mechanism, not an incompatible alternative.

C. Pedagogical Decoupling as Validation

TDL's central contribution—the decoupling of instructional methodology from domain content—serves as a validation case for ADL 2.0's extensibility.

1) The Decoupling Pattern: TDL separates three concerns that ADL 1.0 conflated:

1) **How to execute**: The Engine (shared across all TDL tutors)
2) **How to teach**: The Instructional Model (reusable across courses)
3) **What to teach**: The Content Source (course-specific)

This separation allows a single instructional methodology (e.g., Bloom 8-Step Interactive) to be applied to courses in Law, Programming, Biology, or any other domain without modification.

2) ADL 2.0 Enables This Pattern: ADL 2.0's architectural features directly enable TDL's decoupling:

- **Core/Profile separation**: Allows TDL to define educational abstractions (Instructional Model, Learning Sequence) as profile extensions without modifying the core.
- **Declarative inheritance**: Enables learning sequences to inherit from instructional models cleanly.
- **External references**: Supports the Content Source layer as a separate, updatable resource.

The fact that TDL's pedagogical decoupling can be expressed within ADL 2.0's framework validates the claim that ADL 2.0 provides a sufficiently flexible foundation for domain-specific specializations.

D. Implications for Other Domains

TDL demonstrates a pattern that other domains can follow: identify domain-specific abstractions, implement them as profile extensions, and leverage ADL 2.0's core mechanisms for shared concerns.

1) Potential Domain Profiles: The TDL pattern suggests similar profiles for other domains:

- **Healthcare**: Decoupling clinical protocol (methodology) from patient data (content)
- **Legal**: Decoupling argumentation style from case specifics
- **Customer Service**: Decoupling interaction patterns from product knowledge

Each domain would define its own layers analogous to TDL's Instructional Model and Content Source, while inheriting ADL 2.0's core infrastructure.

2) The Profile Ecosystem: This analysis supports the vision of an ADL 2.0 profile ecosystem where:

1) Domain experts identify reusable patterns in their field
2) These patterns are formalized as ADL 2.0 profiles
3) Practitioners specialize profiles for their specific use cases
4) The community shares and refines validated profiles

TDL is the first complete example of this ecosystem pattern, demonstrating its viability.

E. Bidirectional Validation

The relationship between TDL and ADL 2.0 provides bidirectional validation:

- **TDL validates ADL 2.0**: The fact that TDL's complex pedagogical decoupling can be expressed as an ADL 2.0 profile confirms that ADL 2.0's extensibility mechanisms are sufficient for real-world domain specializations.
- **ADL 2.0 validates TDL**: The alignment with ADL 2.0's principled architecture confirms that TDL's design decisions were not ad-hoc solutions but instances of sound software engineering principles (separation of concerns, inheritance, modularity).

This bidirectional relationship strengthens confidence in both specifications and supports their adoption by the respective communities.

IX. Discussion

A. What Does TDL Contribute?

TDL does not introduce novel pedagogical concepts. Its contribution is one of **pragmatic implementation**: it translates established principles of instructional design into a format that works on commercial LLMs without requiring additional infrastructure.

Specifically, TDL offers:

- **Immediate portability**: The same files work on ChatGPT, Claude, Gemini, and OpenWebUI.
- **Accessible format**: YAML is more readable than XML for non-technical users.
- **Demonstrated reusability**: An instructional model can be applied to multiple courses from different disciplines.
- **Formal specification**: JSON Schemas enable automatic validation.
- **Theory alignment**: Instructional events correspond to established frameworks (Gagné, Bloom).

B. Comparison with Related Work

Regarding other prompt structuring languages:

PDL (IBM): Offers sophisticated technical capabilities (type system, function calls) but is oriented toward developers, not educators. It does not incorporate instructional design concepts.

POML (Microsoft): Its XML syntax provides great expressiveness but introduces unnecessary complexity for the educational use case. The CSS-like style system is powerful but distant from educators' language.

IMS Learning Design: Conceptually complete but failed in adoption due to ecosystem problems, not conceptual complexity. TDL attempts to learn from this failure by prioritizing simplicity and portability.

TDL occupies a specific niche: it combines YAML's readability with concepts specific to instructional design (events, pedagogical models, learning sequences) that neither PDL nor POML contemplate.

C. Implications for Instructional Design Practice

TDL's architecture suggests a natural distribution of professional roles:

- **Senior instructional designer**: Creates and validates reusable instructional models. Requires deep knowledge of learning theories.
- **Teacher / Course designer**: Creates learning sequences applying existing models. Requires content knowledge and basic YAML familiarity.
- **Content expert**: Creates and updates source content. Only requires subject matter expertise.

This separation enables efficient collaboration: an instructional designer can create a "Problem-Based Learning" model that teachers in Medicine, Engineering, and Law then use, each with their specific content.

D. Ethical Considerations

TDL keeps the teacher as author and ultimate responsible party for the educational process. The LLM executes what the teacher has specified, preserving pedagogical authorship. YAML's transparency allows any observer to inspect exactly what methodology the tutor follows.

However, concerns persist:

- **LLM biases**: The underlying model may introduce biases not specified in TDL.
- **Technological dependency**: Students may develop expectations of continuous availability that human tutors cannot satisfy.
- **Privacy**: Conversations with the tutor are processed on third-party servers (OpenAI, Anthropic, Google).

E. Limitations

We acknowledge several limitations of TDL in its current state:

Dependence on the underlying LLM: TDL assumes the LLM will faithfully follow Engine instructions and prompts. In practice, models may occasionally deviate, especially in long conversations.

Learning curve for YAML: Although YAML is more readable than other formats, it still requires attention to indentation and syntax. A visual editor that generates YAML automatically would facilitate adoption.

Improvised pedagogies: TDL is optimized for structured methodologies with predefined sequences. Highly improvised or emergent approaches may not benefit as much from formalization.

Absence of empirical evaluation: This paper presents TDL as architecture and specification. Empirical validation with real students is planned but not yet executed.

Absence of student model: TDL does not implement a student model. There is no cognitive diagnosis, misconception tracking, or adaptation based on learner history.

F. Risks Inherited from IMS LD

The history of IMS Learning Design suggests caution. Despite its conceptual soundness, IMS LD did not achieve widespread adoption. TDL mitigates some of the identified risks:

- **Ecosystem**: TDL does not require special tools; any text editor suffices.
- **Portability**: TDL works on multiple platforms without modification.
- **Terminology**: TDL uses familiar terms (events, prompts, units).

However, TDL inherits the fundamental risk that the formalization effort may not be perceived as proportional to the benefit obtained. Only empirical validation will determine whether teachers adopt TDL in practice.

G. What TDL Is Not

It is important to reiterate what TDL does not claim to be:

- TDL **is not a complete ITS**: It lacks the diagnostic and adaptation components that define traditional ITS.
- TDL **does not guarantee effectiveness**: A poorly designed TDL tutor will be as ineffective as any other poorly designed instruction.
- TDL **does not replace the teacher**: The teacher remains the author of pedagogical design; TDL only formalizes and scales their expertise.
- TDL **does not solve LLM limitations**: If the underlying model cannot follow complex instructions, TDL cannot compensate.

X. Conclusions and Future Work

A. Summary of Contributions

This paper has presented TDL (Tutor Description Language), an evolution of ADL that implements a four-layer architecture for LLM-based tutoring systems. The main contributions are:

1) **Decoupled architecture**: Separation of Engine, Instructional Model, Learning Sequence, and Content Source, enabling independent evolution and reuse.
2) **Instructional model as explicit component**: Formalization of "how to teach" as a sequence of reusable instructional events, aligned with Gagné and Bloom theories.
3) **Methodology-content separation**: The same instructional model can be applied to courses from different disciplines; the same content can be taught with different methodologies.
4) **Documented evolution from ADL 1.0**: Analysis of ADL 1.0 limitations for tutoring and design principles that guided TDL.
5) **Formal specification and tools**: JSON Schema schemas and Python validator to ensure correctness before deployment.
6) **Cross-platform portability**: The same TDL files work on ChatGPT, Claude, Gemini, and OpenWebUI without modification.
7) **Alignment with ADL 2.0**: Demonstration that TDL can be expressed as an ADL 2.0 profile, validating both the chronological development path and ADL 2.0's extensibility mechanisms.

TDL does not claim to be conceptually novel, but pragmatically useful: an accessible implementation of established principles, adapted to the reality of current LLMs.

B. Research Agenda

Empirical validation of TDL requires answering specific research questions:

- **RQ1**: Do TDL tutors produce greater learning gains than LLMs without formal pedagogical structure?
- **RQ2**: Does TDL reduce teaching load while maintaining or improving the quality of individualized attention?
- **RQ3**: Can teachers without advanced technical training create functional TDL tutors in reasonable time?

- **RQ4**: Which instructional models are most effective for which types of content and student population?

C. Hypotheses

Based on the reviewed literature, we propose the following hypotheses:

- **H1**: TDL tutors with the Bloom 8-Step model will produce greater learning gains than LLMs with generic prompts, for technical content.
- **H2**: Teachers with deployed TDL tutors will report less time spent on repetitive individualized attention.
- **H3**: Teachers without prior experience will achieve functional TDL tutors in less than 2 hours of work.
- **H4**: Interactive models will be preferred for conceptual content; expository models for normative content.

D. Proposed Experimental Design

We plan a quasi-experimental study during the 2025-2026 academic year with three conditions:

1) **TDL group**: Students with access to a TDL tutor using the Bloom 8-Step model.
2) **Generic LLM group**: Students with access to ChatGPT/Claude with a basic prompt.
3) **Control group**: Students with traditional materials (notes, videos).

Metrics:

- Normalized gain pre-post on knowledge tests.
- Self-reported study time.
- Engagement (number and depth of interactions with the tutor).
- Student satisfaction (Likert scale).
- Teaching load (hours dedicated to individualized attention).

E. Future Work

We identify several directions for future work:

Visual editor (TDL Maker): Development of a web tool that allows designing learning sequences visually, automatically generating the YAML. This would eliminate the syntactic barrier for teachers without technical experience.

Learning analytics: Integration of hooks to record which events are executed, student response times, interaction patterns, and dropout points. This data would inform iterative improvement of models and sequences.

Community repository: Creation of an open repository of validated instructional models, where designers can share methodologies and teachers can discover models appropriate for their needs.

Assessment extensions: Adding sections for evaluation rubrics, grading criteria, and automatic generation of student progress reports.

Lightweight student model: Exploring the feasibility of incorporating a simplified student model, leveraging LLMs' conversational memory or external storage, to enable some history-based adaptation.

LMS integration: Developing connectors to integrate TDL tutors with learning management systems (Moodle, Canvas), enabling authentication, progress tracking, and grade synchronization.

ADL 2.0 profile formalization: Completing the formal definition of TDL as an ADL 2.0 profile, including schema definitions and validation rules that ensure compliance with both specifications.

F. Final Conclusion

The promise of scaling personalized education through AI will not be fulfilled solely with more powerful language models. It requires methods for educators to transfer their pedagogical expertise to these systems. TDL offers a pragmatic path toward this goal: it allows "how to teach" to be formalized as reusable knowledge while "what to teach" remains under the teacher's control.

By separating responsibilities into well-defined layers and aligning with established instructional design theories, TDL facilitates collaboration among instructional designers, teachers, and content experts. The result is a framework that amplifies educators' impact without compromising their pedagogical authorship.

The alignment with ADL 2.0 validates TDL's architectural decisions as instances of sound software engineering principles, while simultaneously demonstrating that ADL 2.0's extensibility mechanisms are sufficient for real-world domain specializations. This bidirectional validation strengthens confidence in both specifications.

Empirical validation will determine whether the lessons of the past (particularly from IMS Learning Design) have been learned. TDL is a proposal in that direction, subject to experimental scrutiny and iterative improvement.

ACKNOWLEDGMENTS

This work was partially funded by the project AI4ProSa.

REFERENCES

[1] B. S. Bloom, "The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring," *Educational Researcher*, vol. 13, no. 6, pp. 4–16, 1984.
[2] K. VanLehn, "The relative effectiveness of human tutoring, intelligent tutoring systems, and other tutoring systems," *Educational Psychologist*, vol. 46, no. 4, pp. 197–221, 2011.
[3] J. A. Kulik and J. D. Fletcher, "Effectiveness of intelligent tutoring systems: A meta-analytic review," *Review of Educational Research*, vol. 86, no. 1, pp. 42–78, 2016.
[4] W. Ma, O. O. Adesope, J. C. Nesbit, and Q. Liu, "Intelligent tutoring systems and learning outcomes: A meta-analysis," *Journal of Educational Psychology*, vol. 106, no. 4, pp. 901–918, 2014.
[5] D. Lee et al., "The impact of generative AI on higher education learning and teaching: A study of educators' perspectives," *Computers and Education: Artificial Intelligence*, vol. 6, p. 100221, 2024.
[6] G. Kestin et al., "AI tutoring outperforms active learning," *Nature Human Behaviour*, 2025.
[7] A. Tack and C. Piech, "The AI teacher test: Measuring the pedagogical ability of blender and GPT-3 in educational dialogues," in *Proc. 15th Int. Conf. on Educational Data Mining*, 2022.
[8] J. Macina et al., "Opportunities and challenges in neural dialog tutoring," in *Proc. 16th Int. Conf. on Educational Data Mining*, 2023.
[9] C. Borchers et al., "Comparing LLM-based tutors to experienced human tutors," in *Proc. Int. Conf. on Artificial Intelligence in Education (AIED)*, 2025.

[10] R. Puech et al., "Towards pedagogical steering of LLMs for tutoring: A case with productive failure," *arXiv preprint arXiv:2410.03781*, 2024.
[11] A. Scarlatos et al., "Exploring the limitations of LLMs as intelligent tutoring systems," *arXiv preprint arXiv:2504.05570*, 2025.
[12] H. S. Nwana, "Intelligent tutoring systems: An overview," *Artificial Intelligence Review*, vol. 4, no. 4, pp. 251–277, 1990.
[13] B. P. Woolf, *Building Intelligent Interactive Tutors*. Burlington, MA: Morgan Kaufmann, 2009.
[14] J. R. Anderson et al., "Cognitive tutors: Lessons learned," *The Journal of the Learning Sciences*, vol. 4, no. 2, pp. 167–207, 1995.
[15] A. T. Corbett and J. R. Anderson, "Knowledge tracing," *User Modeling and User-Adapted Interaction*, vol. 4, no. 4, pp. 253–278, 1994.
[16] V. Aleven et al., "A new paradigm for intelligent tutoring systems: Example-tracing tutors," *Int. Journal of Artificial Intelligence in Education*, vol. 19, no. 2, pp. 105–154, 2009.
[17] T. Murray, "Authoring intelligent tutoring systems: An analysis of the state of the art," *Int. Journal of Artificial Intelligence in Education*, vol. 10, pp. 98–129, 1999.
[18] R. A. Sottilare et al., "Design recommendations for intelligent tutoring systems," U.S. Army Research Laboratory, 2012.
[19] A. C. Graesser et al., "AutoTutor: An intelligent tutoring system with mixed-initiative dialogue," *IEEE Transactions on Education*, vol. 48, no. 4, pp. 612–618, 2004.
[20] Z. A. Pardos and S. Bhandari, "Learning gain differences between ChatGPT and human tutors-generated hints," *PLOS ONE*, vol. 19, no. 3, e0300722, 2024.
[21] M. D. Merrill, "Component display theory," in *Instructional-Design Theories and Models*, C. M. Reigeluth, Ed. Hillsdale, NJ: Lawrence Erlbaum, 1983, pp. 279–333.
[22] M. D. Merrill, L. Li, and M. K. Jones, "Instructional transaction theory: An introduction," *Educational Technology*, vol. 31, no. 6, pp. 7–12, 1991.
[23] R. M. Gagne, *The Conditions of Learning and Theory of Instruction*, 4th ed. New York: Holt, Rinehart and Winston, 1985.
[24] L. W. Anderson and D. R. Krathwohl (Eds.), *A Taxonomy for Learning, Teaching, and Assessing: A Revision of Bloom's Taxonomy of Educational Objectives*. New York: Longman, 2001.
[25] R. Koper and C. Tattersall (Eds.), *Learning Design: A Handbook on Modelling and Delivering Networked Education and Training*. Berlin: Springer, 2005.
[26] M. Derntl et al., "The conceptual structure of IMS Learning Design does not impede its use for authoring," *IEEE Transactions on Learning Technologies*, vol. 5, no. 1, pp. 74–86, 2012.
[27] D. Griffiths et al., "Learning design tools," in *Learning Design*, R. Koper and C. Tattersall, Eds. Berlin: Springer, 2005, pp. 109–135.
[28] A. Berggren et al., "Practical and pedagogical issues for teacher adoption of IMS Learning Design," *Journal of Interactive Media in Education*, vol. 2005, no. 1, 2005.
[29] S. Neumann and R. Oberhuemer, "User evaluation of a graphical modeling tool for IMS Learning Design," in *Proc. 8th Int. Conf. on Advanced Learning Technologies*, IEEE, 2008, pp. 287–291.
[30] S. E. Ainsworth et al., "Using edit distance algorithms to compare alternative approaches to ITS authoring," in *Proc. Int. Conf. on Intelligent Tutoring Systems*, Springer, 2002.
[31] A. van Deursen et al., "Domain-specific languages: An annotated bibliography," *ACM SIGPLAN Notices*, vol. 35, no. 6, pp. 26–36, 2000.
[32] M. Fowler, *Domain-Specific Languages*. Boston: Addison-Wesley, 2010.
[33] IBM Research, "Prompt Declaration Language (PDL)," 2024. [Online]. Available: https://ibm.github.io/prompt-declaration-language/
[34] Y. Zhang, N. Chen, J. Xu, and Y. Yang, "Prompt Orchestration Markup Language," *arXiv preprint arXiv:2508.13948*, 2025.
[35] P. Pernias et al., "ADL-based educational assistant architecture: Transforming pedagogical design into automated tutoring systems," in *Proc. ICERI 2025*, 2025.
[36] P. A. Pernías Peco and M. P. Escobar Esteban, "ADL 2.0: A core specification framework for LLM-based assistants," *arXiv preprint*, 2025.
[37] J. White et al., "A prompt pattern catalog to enhance prompt engineering with ChatGPT," *arXiv preprint arXiv:2302.11382*, 2023.
[38] L. Reynolds and K. McDonell, "Prompt programming for large language models: Beyond the few-shot paradigm," in *CHI 2021 Extended Abstracts*. ACM, 2021, pp. 1–7.
[39] W. Holmes, M. Bialik, and C. Fadel, *Artificial Intelligence in Education: Promises and Implications for Teaching and Learning*. Boston: Center for Curriculum Redesign, 2019.