Specification-Guided Synthesis of Deadlock-Free Communication Protocol Refinements with Large Language Models

Yang Li
University of Oxford
Computer Science
Oxford, United Kingdom
yang.li@wolfson.ox.ac.uk

Ping Hou
University of Oxford
Computer Science
Oxford, United Kingdom
ping.hou@cs.ox.ac.uk

Nobuko Yoshida
University of Oxford
Computer Science
Oxford, United Kingdom
nobuko.yoshida@cs.ox.ac.uk

Abstract

Ensuring *behavioural correctness* in *communication protocols* is a central challenge in distributed software systems, as subtle inconsistencies can lead to deadlocks. In such settings, *protocol refinement* – the safe substitution of a protocol that preserves correctness and compatibility with other components – is essential. Large language models (LLMs) have demonstrated strong capabilities in code generation and program synthesis, yet lack mechanisms to reliably produce outputs with correct behaviour. Formal specification approaches, such as *multiparty session types* (MPST), offer rigorous guarantees, including deadlock freedom, but provide limited support for automatically constructing protocol refinements. In this paper, we present Syntropy, a framework for synthesising protocol refinements guided by MPST specifications and LLMs. It incorporates refinement constraints directly into the generation process, ensuring the generated variants satisfy these guarantees. Our comprehensive evaluation indicates that Syntropy achieves 95.6%–99.5% validity while maintaining high syntactic correctness, and produces diverse, non-trivial refinements across multiple LLMs.

CCS Concepts

• **Software and its engineering** → **Software notations and tools**; • **Networks** → **Network protocols**; • **Computing methodologies** → *Artificial intelligence*.

Keywords

formal specifications, large language models, protocol refinement, behavioural correctness, constrained generation, session types

1 Introduction

Modern software engineering is increasingly shaped by large language models (LLMs) [48], which are widely used to automate programming tasks such as code generation [9], transformation [43], and program synthesis [2]. Recent work [35, 36, 43] shows that LLMs can produce syntactically correct and functionally useful code across diverse tasks, including programs that satisfy certain semantic properties such as type safety, indicating their potential to support complex software development workflows.

Despite these advances, a key limitation remains: LLM-generated artefacts do not provide behavioural guarantees by construction, especially in systems with complex interactions. This is particularly critical in distributed software systems, from microservices [17] to cyber-physical systems [28], where reliability depends on consistent communication between components as well as local functionality. Subtle interaction errors, such as incorrect message ordering or unintended cyclic dependencies, can lead to failures, including *deadlocks*. Ensuring that LLM-generated or modified programs preserve reliable communication remains an open challenge.

Such systems rely on *communication protocols* to coordinate independently developed components. As systems evolve, protocols must be adapted to incorporate new functionality or modify interaction patterns. Even minor changes can have system-wide effects that are difficult to predict and diagnose. *Protocol refinement* – replacing a protocol with a behaviourally compatible alternative – offers a principled approach to managing this evolution. Nevertheless, preserving the intended interaction properties during this process is non-trivial in practice. In many cases, refinements are constructed manually, making them error-prone and hard to validate. This motivates the following research question:

> *How can protocol refinements be systematically synthesised while retaining behavioural correctness?*

To address this question, we draw on specification techniques such as session types [23], in particular multiparty session types (MPST) [24, 44], which are well established for describing communication protocols and reasoning about their behaviour. MPST captures global interaction structures and enforcs properties such as communication safety and deadlock freedom. Toolchains for MPST have been developed for over 30 programming languages [52], facilitating its adoption in software engineering.

MPST further defines refinement relations that allow protocols to be transformed into more flexible forms while preserving these properties [21, 22]. Although checking such relations can be automated [5–7, 15], generating variants that satisfy refinement constraints is not straightforward and, in some settings, *undecidable* [8, 27], limiting the automation of protocol refinement.

To bridge this gap, we propose Syntropy, the first framework combining LLMs with MPST to support the synthesis of protocol refinements, to the best of our knowledge. Given an MPST protocol, Syntropy constructs protocol candidates based on structured representations of interaction behaviour and refinement objectives. The generation process is coupled with validation against refinement constraints, ensuring that only valid outputs are retained. This approach extends LLM-based generation beyond syntactic and local semantic aspects by incorporating protocol-level requirements, including deadlock freedom. By integrating generative models with specification-based verification, Syntropy explores the space of valid protocol refinements and produces varied and reliable alternatives that are difficult to construct manually.

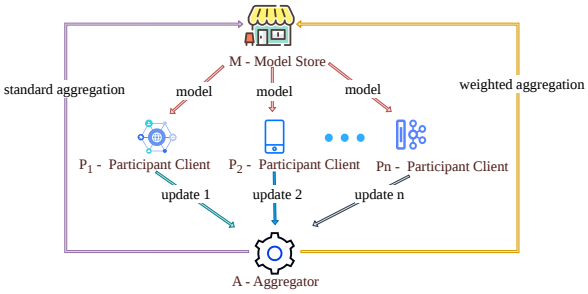
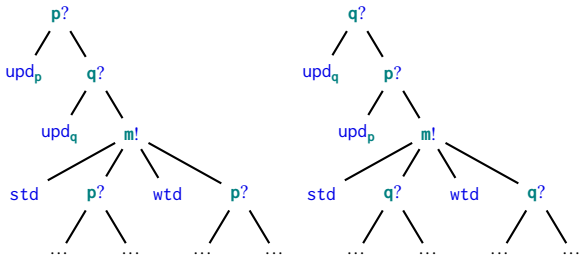
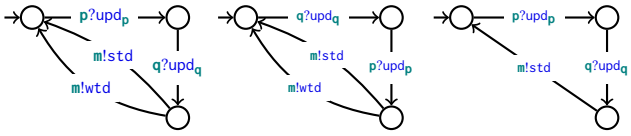


Figure 1: Federated learning protocol



(a) Session tree $\mathbb{T}_a$ for T_a **(b) Session tree $\mathbb{T}'_a$ for T'_a**

Figure 2: Session trees



(a) FSM for a **(b) Reordered FSM for a** **(c) Covariant FSM for a**

Figure 3: FSMs for role a

We evaluate Syntropy on two datasets, comprising protocols derived from the literature and synthetic benchmarks, using multiple LLMs of varying sizes: three 7B code models, a general-purpose 7B model, and a 32B model. Syntropy attains 95.6%–99.5% validity across all models, while maintaining strong syntactic correctness (95.4%–98.1%). Furthermore, it produces multiple distinct refinements for each specification, demonstrating its ability to identify alternative protocol formulations, rather than trivial variations. Ablation studies indicate that both specification guidance and constraint-based validation are essential for attaining high validity and diversity.

We additionally compare Syntropy with frontier language models, demonstrating that, although frontier language models can synthesise valid protocol refinements, their coverage across protocol specifications remains limited. This highlights the need for the proposed approach, which enables comprehensive protocol refinement while supporting reproducible evaluation and local deployment with fine-tuned open models.

The contributions of this paper are as follows:

- **LLM Generation with Behavioural Guarantees.** We propose a novel approach to enable LLMs to synthesise MPST protocol refinements with guaranteed behavioural correctness.
- **Specification-Guided Protocol Refinement.** We introduce a systematic encoding of MPST specifications that guides and constrains LLM-based generation of protocol refinements.
- **Constraint-Integrated Generation.** We design a two-level generation workflow that incorporates constraint validation into the synthesis via prefix filtering and subsequent verification.
- **Syntropy Framework and Empirical Evaluation.** We implement Syntropy and evaluate it across multiple LLMs and datasets, demonstrating high validity and generating structurally distinct and non-superficial protocol refinements.

2 Motivation and Background

Motivation. Fig. 1 depicts the message exchange coordinating distributed model training, referred to as the *federated learning protocol*, reflecting communication patterns in centralised federated learning systems [26]. The protocol involves a model store m, clients $\mathtt{p}_1, \ldots, \mathtt{p}_n$, and an aggregator a, and proceeds in phases as follows: (1) *Model Distribution*: the model store sends the current global model to all clients; (2) *Update Submission*: each client performs local computation and sends an update to the aggregator; and (3) *Aggregation Result*: the aggregator combines the updates and sends either a standard or weighted result to the model store. The model store then updates the global model and redistributes it for the next round.

We consider a minimal instance of the federated learning protocol with two clients p and q, enforcing an ordered submission of updates to the aggregator a, where p precedes q.

From the perspective of a under *synchronous* communication, the local protocol can be represented by a finite state machine (FSM), as shown in Fig. 3a, with the following steps: (1) receive (?) $\mathtt{upd}_\mathtt{p}$ from p; (2) receive $\mathtt{upd}_\mathtt{q}$ from q; (3) send (!) aggregated result – either std or wtd – to the model store m; and (4) repeat from step 1.

In practice, communication is *asynchronous* and typically modelled as FIFO message passing. In this scenario, a may receive the update from q before that from p, violating the ordering in Fig. 3a and necessitating *refinement* to account for *message reordering*. The refined FSM, shown in Fig. 3b, captures this by admitting the reception order q followed by p, while preserving the subsequent aggregation and response behaviour. As the updates from p and q are independent, the two FSMs are refinements of each other.

Furthermore, additional refinements are possible. The general protocol in Fig. 1 can be viewed as a *contravariant* refinement of the minimal instance, as it admits a broader class of inputs, while *covariant* refinement arises when the aggregator deterministically produces a single fixed result (Fig. 3c).

Such refinements occur naturally in distributed software systems, where communication patterns are commonly adapted for efficiency, e.g. a service may omit certain responses to reduce latency. Nevertheless, even minor changes may introduce severe communication inconsistencies. For example, refining the behaviour of a to exclude updates from q, while leaving other participants unaffected, yields a communication system that violates compatibility, leading

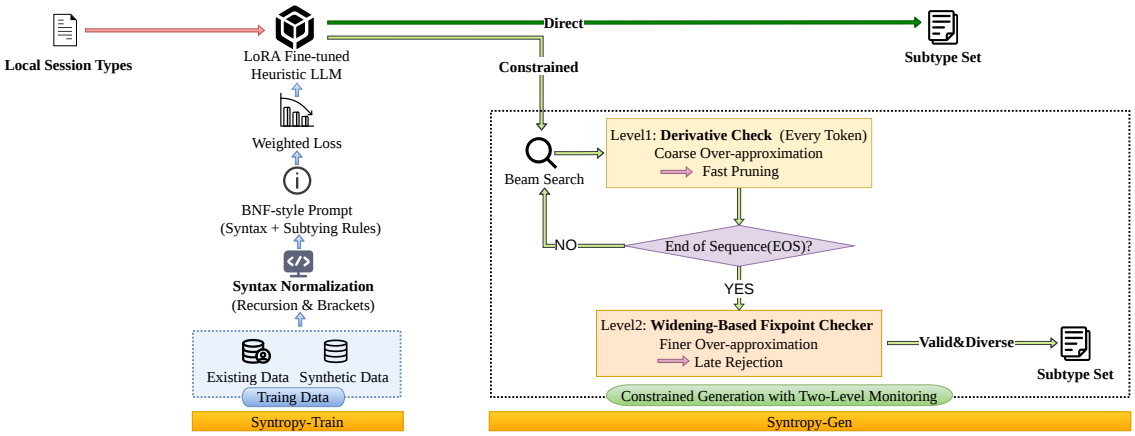


Figure 4: Overview of Syntropy

to deadlock. This highlights the challenge of designing correct refinements and motivates the application of formal specifications to ensure the rigorous synthesis of protocol refinements that preserve communication properties.

MPST and Asynchronous Subtyping. Multiparty Session Types (MPST) [24, 44] provide a framework for specifying and verifying communication protocols. In MPST, the communication behaviour of each role (denoted by $\mathtt{p}, \mathtt{q}, \mathtt{s}, \mathtt{p}', \ldots \in \mathcal{R}$) is described by *local session types* (or simply *session types*), and protocol refinement is formalised as a subtyping relation on these types. In particular, *asynchronous multiparty subtyping* (AMS) [22] offers a sound and complete declarative characterisation of this relation, ensuring that subtypes – i.e. type-based protocol refinements – can safely replace their supertypes while preserving all desirable communication safety properties.

The syntax of session types, ranging over $T, T', T_i, \ldots$, is inductively defined as follows:

$$T ::= \oplus_{i\in I}\, \mathtt{p}!\mathtt{m}_i.T_i \;\mid\; \&_{i\in I}\, \mathtt{p}?\mathtt{m}_i.T_i \;\mid\; \mu \mathtt{t}.T \;\mid\; \mathtt{t} \;\mid\; \mathbf{end}$$

The constructs $\oplus_{i\in I}\, \mathtt{p}!\mathtt{m}_i.T_i$ and $\&_{i\in I}\, \mathtt{p}?\mathtt{m}_i.T_i$ denote *internal* and *external choices*, corresponding to sending (!) to or receiving (?) from role $\mathtt{p}$ a label $\mathtt{m}_i$, respectively, where $I \neq \emptyset$ and the labels $\mathtt{m}_i$ are pairwise distinct. The type **end** marks *termination* (omitted when unambiguous), while $\mu \mathtt{t}.T$ introduces recursion with variable $\mathtt{t}$.

The behaviour of the aggregator $\mathtt{a}$ in the minimal federated learning protocol is captured by

$$T_{\mathtt{a}} = \mu \mathtt{t}.\mathtt{p}?\mathsf{upd}_{\mathtt{p}}.\mathtt{q}?\mathsf{upd}_{\mathtt{q}}.\oplus\{\mathtt{m}!\mathtt{std}.\mathtt{t}, \mathtt{m}!\mathtt{wtd}.\mathtt{t}\},$$

which is an alternative representation of the FSM in Fig. 3a.

In AMS, besides covariance (fewer output options) (**C1**) and contravariance (more input options) (**C2**), two forms of asynchronous message reordering are allowed, whereby a subtype may anticipate input and output actions of the supertype:

R1. An input from $\mathtt{p}$ may be anticipated before a finite number of inputs not from $\mathtt{p}$;

R2. An output to $\mathtt{p}$ may be anticipated before a finite number of inputs (from any participant), and before outputs not directed to $\mathtt{p}$.

To formalise AMS, session types are interpreted as (possibly infinite) session trees. For instance, Fig. 2a presents the session tree $\mathbb{T}_{\mathtt{a}}$ associated with $T_{\mathtt{a}}$. A coinductive tree refinement relation $\lesssim$ is defined on such session trees, and the asynchronous subtyping relation $\leqslant$ is obtained by lifting $\lesssim$ to session types.

Consider the session type (equivalent to FSM in Fig. 3b)

$$T'_{\mathtt{a}} = \mu \mathtt{t}.\mathtt{q}?\mathsf{upd}_{\mathtt{q}}.\mathtt{p}?\mathsf{upd}_{\mathtt{p}}.\oplus\{\mathtt{m}!\mathtt{std}.\mathtt{t}, \mathtt{m}!\mathtt{wtd}.\mathtt{t}\},$$

obtained from $T_{\mathtt{a}}$ by reordering the inputs. Its session tree $\mathbb{T}'_{\mathtt{a}}$, shown in Fig. 2b, is related to $\mathbb{T}_{\mathtt{a}}$ by $\lesssim$; hence $T'_{\mathtt{a}} \leqslant T_{\mathtt{a}}$, indicating that $T'_{\mathtt{a}}$ can safely substitute for $T_{\mathtt{a}}$.

AMS is undecidable in general [8, 27]. Together with its intrinsic complexity, this renders the synthesis of asynchronous subtypes substantially non-trivial and inherently error-prone, giving rise to the main research question of this paper:

> *How can subtypes be synthesised automatically under asynchronous multiparty subtyping?*

To tackle this challenge, we propose a system based on large language models, described in the next section.

3 Syntropy: A Framework for Asynchronous Subtype Synthesis

Figure 4 illustrates the overall architecture of Syntropy, a framework for automatically synthesising asynchronous subtypes from a source session type. It consists of two complementary modules: Syntropy-Train, which trains large language models (LLMs) to learn subtype generation patterns, and Syntropy-Gen, which leverages the trained model to produce diverse asynchronous subtypes guided by the asynchronous multiparty subtyping introduced in §2, thereby improving syntactic and semantic consistency.

3.1 Syntropy-Train: Learning Asynchronous Subtype Generation

We fine-tune an open-source model (e.g. Qwen2.5-Coder-7B-Instruct [25]) using LoRA to generate subtypes conditioned on a given session type (the *supertype*). Since a supertype may correspond to an unbounded number of subtypes under asynchronous subtyping, the

Table 1: Transformations encoded in prompts

Rule	Prompt	Example
Identity	Subtype identical to supertype.	–
Unfold	Unfold recursion via substitution of the recursive variable.	Valid REC_X_OPEN p?m; p!m′; REC_X_CLOSE ⇒ p?m; p!m′; REC_X_OPEN p?m; p!m′; REC_X_CLOSE
RefA	Move recv p?m earlier when roles differ.	Valid p?m; q?m′; $T \leqslant$ q?m′; p?m; T Invalid p?m; p?m′; $T \leqslant$ p?m′; p?m; T
RefB	Move send p!m earlier across independent actions.	Valid p!m; q?m′; $T \leqslant$ q?m′; p!m; T Invalid p!m; p!m′; $T \leqslant$ p!m′; p!m; T
RefOut	Internal choice: subtype offers fewer labels.	Valid lbrace p!m; T_1 rbrace $\leqslant$ lbrace p!m; T_1, p!m′; T_2 rbrace Invalid lbrace p!m; T_1, p!m′; T_3 rbrace $\leqslant$ lbrace p!m; T_1, p!m′; T_2 rbrace
RefIn	External choice: subtype accepts more labels.	Valid lbrace p?m_1; T_1, p?m_2; T_2, p?m_3; T_3 rbrace $\leqslant$ lbrace p?m_1; T_1, p?m_2; T_2 rbrace Invalid lbrace p?m_1; T_1 rbrace $\leqslant$ lbrace p?m_1; T_1, p?m_2; T_2 rbrace

model is trained to capture structure-preserving transformations and generate diverse and structurally distinct, valid outputs.

Our training data consists of session type specifications derived from MPST literature [5–7, 15], augmented with synthetically generated examples to improve coverage and diversity (see §4.1.1 for details). Each instance is of the form $\{S, [\{T_1, 1_1\}, \ldots, \{T_n, 1_n\}], n\}$, where S denotes the supertype, n the number of subtypes, and for each $i \leq n$, T_i is a subtype with auxiliary (heuristic) label 1_i (e.g. indicating one or multiple transformations). To accommodate variable-length instances, we employ dynamic padding during training for efficient batching with minimal overhead. This design exposes the model to diverse transformation patterns, allowing it to generate valid subtypes beyond a fixed set of explicitly defined rules.

To facilitate training, we adapt the syntax of session types in §2 into a model-friendly representation:

$$\begin{aligned} T \quad ::= \quad & \mathbf{p}!\mathtt{m}; T \mid \mathbf{p}?\mathtt{m}; T \mid \mathbf{end} \\ \mid \quad & \mathtt{lbrace}\ \mathbf{p}!\mathtt{m}_1; T_1, \ldots, \mathbf{p}!\mathtt{m}_n; T_n\ \mathtt{rbrace} \\ \mid \quad & \mathtt{lbrace}\ \mathbf{p}?\mathtt{m}_1; T_1, \ldots, \mathbf{p}?\mathtt{m}_n; T_n\ \mathtt{rbrace} \\ \mid \quad & \mathtt{REC\_X\_OPEN}\ T \mid \mathtt{REC\_X\_CLOSE} \end{aligned}$$

Send and receive actions are explicitly represented in sequential form as $\mathbf{p}!\mathtt{m}; T$ and $\mathbf{p}?\mathtt{m}; T$, yielding a well-defined action structure suitable for sequence modelling; internal and external choices use $\mathtt{lbrace} \cdot \mathtt{rbrace}$; recursion is expressed by $\mathtt{REC\_X\_OPEN}\ T$, binding $\mathbf{X}$ over T, with recursive occurrences encoded as $\mathtt{REC\_X\_CLOSE}$, each denoting a continuation to the corresponding binder. These modifications preserve semantics while making structural boundaries explicit and reducing syntactic ambiguity.

For each training instance, we construct a prompt consisting of (1) a theoretical context, including transformation descriptions (e.g. Identity, RefA, RefB, RefIn, RefOut, and Unfold), and (2) the instance-specific input. The theoretical context is derived from the prompt components summarised in Table 1 and provides structured guidance for subtype generation. RefA and RefB correspond to the message reordering rules **R1** and **R2** in §2, respectively, while RefOut and RefIn capture covariance (**C1**) and contravariance (**C2**). Additionally, Identity instantiates the reflexivity of subtyping, ensuring the existence of at least one valid subtype for every supertype, namely itself, and Unfold expands recursive types by recursively substituting bound variables.

Example 3.1 (Multiple Transformations). Consider the supertype:

$S = \mathbf{p}?\mathtt{m}_1; \mathtt{lbrace}\ \mathbf{p}!\mathtt{m}_2; \mathbf{q}?\mathtt{m}_3; \mathbf{r}?\mathtt{m}_4; \mathbf{end}, \mathbf{p}!\mathtt{m}_5; \mathbf{end}\ \mathtt{rbrace}$

Under RefA, the subtype

$T_1 = \mathbf{p}?\mathtt{m}_1; \mathtt{lbrace}\ \mathbf{p}!\mathtt{m}_2; \mathbf{r}?\mathtt{m}_4; \mathbf{q}?\mathtt{m}_3; \mathbf{end}, \mathbf{p}!\mathtt{m}_5; \mathbf{end}\ \mathtt{rbrace}$

is obtained, while applying RefOut yields $T_2 = \mathbf{p}?\mathtt{m}_1; \mathbf{p}!\mathtt{m}_5$.

Note that T_2 admits multiple transformations (e.g. RefA followed by RefOut or directly via RefOut), whereas the generator may assign only one as its heuristic label.

Training sequences follow a standard format consisting of a system prompt, a user prompt, and the corresponding assistant output. Prompt tokens are masked in the loss computation (i.e. excluded from the objective), and loss is computed exclusively over the assistant outputs, corresponding to the generated subtype sequences. This ensures that the theoretical context serves as conditioning information rather than a target for imitation.

We adopt a weighted token-level loss, assigning a weight of 1.0 to subtype tokens and 0.2 to auxiliary fields, including labels and the number of subtypes. Since both may originate from heterogeneous sources (e.g. benchmark data or synthetic generation) and do not necessarily reflect canonical transformation patterns (see Thm. 3.1), they are treated as weak supervision signals. This encourages the model to prioritise learning structurally valid subtype sequences while mitigating overfitting to auxiliary information.

Finally, the model is fine-tuned using LoRA (rank 64, $\alpha = 32$, dropout 0.05) with a maximum sequence length of 8k tokens. Training is performed using AdamW with a cosine learning rate schedule and a warmup ratio of 0.1 for 3 epochs.

3.2 Syntropy-Gen: Synthesising Asynchronous Subtypes

Syntropy-Gen generates asynchronous subtypes of a given supertype using the trained model. It supports two generation strategies: *direct generation*, where the LLM produces subtype candidates guided by transformation rules adopted during training (§ 3.1), and *constrained generation*, which introduces two-level monitoring mechanisms to enforce asynchronous subtyping constraints and improve generation accuracy.

3.2.1 Direct Generation. In the direct generation approach, the trained model is prompted to synthesise subtypes. Generation is guided by transformation patterns – namely Identity, RefA, RefB, RefIn, RefOut, and Unfold– described in § 3.1. These encourage the model to produce diverse candidates reflecting covariance, contravariance, and reordering.

In this process, subtypes are derived from the supertype through sequences of transformations, where successive applications expand the set of generated subtypes. Structured guidance is therefore

Algorithm 1 Constrained Generation with Two-Level Monitoring

Input: supertype S, language model M, beam size k
Output: set of candidate subtypes Ω

$\mathbb{T}_S \leftarrow \textsc{Parse}(S)$
$Beams \leftarrow \{(\epsilon, 0.0)\}$
$C \leftarrow \emptyset$
for each decoding step **do**
  for each $(seq, score) \in Beams$ **do**
    for each $(t, p) \in Top_{2k}(M)$ **do**
      $seq' \leftarrow seq \cdot t$
      $\mathbb{T}' \leftarrow \textsc{ParsePrefix}(seq')$
      ▷ Level 1 (Derivative Check)
      if $\mathbb{T}' \neq \bot$ **and** $\textsc{Feasible}(\mathbb{T}', \mathbb{T}_S)$ **then**
        if t = EOS **then**
          $\mathbb{T}_f \leftarrow \textsc{Parse}(seq')$
          ▷ Level 2 (Widening-Based Fixpoint Check)
          if $\textsc{SimCheck}(\mathbb{T}_f, \mathbb{T}_S)$ **then**
            $\Omega \leftarrow \Omega \cup \{seq'\}$
        else
          $C \leftarrow C \cup \{(seq', score + \log p)\}$
  $Beams \leftarrow Top_k(C)$

provided for the LLM to generate pattern-compatible subtypes; however, semantic correctness is not guaranteed as this naïve approach relies entirely on the trained model.

3.2.2 Constrained Generation with Two-Level Monitoring. To improve the semantic validity of generated subtypes while preserving diversity, we employ constrained generation with two-level monitoring. Our approach is inspired by the asynchronous multiparty subtyping framework of [6], which we adapt to guide both prefix-level filtering and final subtype verification within the constrained generation process.

The input supertype is first parsed into a session tree, a tree-structured representation of a session type, as illustrated in §2. Beam search then explores candidate subtypes, while the monitoring stages enforce semantic constraints throughout generation.

We formalise the constrained generation procedure in Algorithm 1 and describe its main components below.

Level 1: Token-level Derivative Check. Level 1 takes as input a decoded prefix together with the session tree $\mathbb{T}_S$ of the input supertype S, and returns a Boolean indicating whether the prefix can still be extended to a valid subtype of S.

At each decoding step, the prefix is incrementally parsed into a partially constructed session tree $\mathbb{T}'$, whose unresolved positions are represented by `Hole` nodes. Since the MPST grammar is deterministic, each syntactically valid prefix corresponds to a *unique* partial tree, while `Hole` nodes denote subtrees to be completed. The monitor then performs a lightweight coinductive derivative-based compatibility check between $\mathbb{T}_S$ and $\mathbb{T}'$ to determine whether such an extension remains possible. Specifically, it evaluates a predicate $\textsc{Feasible}(\mathbb{T}', \mathbb{T}_S)$, which returns TRUE if $\mathbb{T}'$ admits at least one completion whose corresponding session type is a valid subtype of S, and FALSE otherwise.

The feasibility check is based on derivative reasoning over session trees [6], applied to partial constructions. It establishes whether the behaviour induced by $\mathbb{T}'$ is compatible with that represented by $\mathbb{T}_S$. For `send` nodes, each branch generated in $\mathbb{T}'$ must be supported by a corresponding transition in $\mathbb{T}_S$; for `receive` nodes, realisability is preserved provided that at least one branch of $\mathbb{T}'$ is admitted by $\mathbb{T}_S$, possibly after unfolding recursion in $\mathbb{T}_S$ to expose further input actions. A prefix is rejected only when no completion of the current partial tree can satisfy the subtyping constraints (e.g. because the matched node of $\mathbb{T}_S$ is `end`, i.e. the terminal node).

This monitor yields a prefix-level *over-approximation*: any explored prefix that could lead to a valid subtype is preserved, while some infeasible ones may also be retained. This property is relative to beam search, which does not exhaustively enumerate all paths, and to the generally infinite space of subtypes. In practice, Level 1 serves as an efficient per-token filter within beam search. When $\textsc{Feasible}(\mathbb{T}', \mathbb{T}_S)$ evaluates to FALSE, the corresponding beam is pruned immediately, eliminating infeasible regions of the search space while preserving all potentially valid candidates.

Candidates that reach an end-of-sequence (EOS) token while satisfying the Level 1 condition are forwarded to Level 2; those that produce EOS otherwise are discarded, while all others continue under Level 1 monitoring.

Level 2: Widening-Based Fixpoint Checker. When a candidate reaches an EOS token, it is parsed into a complete session tree $\mathbb{T}_f$ and submitted to a full subtype checker against the input supertype S. The checker, implemented as a function $\textsc{SimCheck}(\mathbb{T}_f, \mathbb{T}_S)$, is built on a derivative-based decision procedure for asynchronous subtyping over session trees [6].

The procedure explores pairs of states from $\mathbb{T}_f$ and $\mathbb{T}_S$, maintaining a worklist of obligations together with the information required to avoid unsound revisiting of recursive configurations. Each obligation is processed according to the structure of the current nodes. For `send` nodes, the action exposed by $\mathbb{T}_f$ must be matched by a corresponding send derivative in $\mathbb{T}_S$; for `receive` nodes, the checker requires that the branching structure of $\mathbb{T}_S$ covers the inputs expected by $\mathbb{T}_f$, in accordance with the asynchronous subtyping conditions. For terminal nodes, the check succeeds only if the corresponding configuration in $\mathbb{T}_S$ admits termination.

To ensure termination in the presence of recursion, the checker applies a *widening operator* at designated recursion points, following the abstract interpretation framework of [6]. Widening merges newly derived configurations with previously explored ones and detects recurring patterns, thereby ensuring convergence of the fixpoint computation. The procedure terminates when the worklist is exhausted. Acceptance then indicates that the session type represented by $\mathbb{T}_f$ is a valid subtype of S.

This checker accepts only candidates satisfying the asynchronous subtyping relation, while some valid ones may be rejected.

Correctness and Design Justification. Each output subtype is required to pass both a coarse prefix-level filter (Level 1) and a finer-grained final verification step (Level 2). This two-level design confines computationally expensive subtype checking to complete candidates, while lightweight prefix-level filtering eliminates infeasible prefixes early during search.

Level 2 operates as a conservative checker rather than a complete decision procedure, reflecting the undecidability of asynchronous subtyping: accepted candidates are valid, whereas rejection does not imply invalidity.

Additionally, this design promotes output diversity by enabling structurally distinct valid subtypes to emerge through different reorderings and interaction patterns permitted by asynchronous subtyping. An overly restrictive validation step would limit the search space explored by beam search and reduce the diversity of generated subtypes. Overall, the two-level design balances semantic correctness with adequate exploration of the candidate space.

Implementation Details. We implement Algorithm 1 using beam search with beam size k. At each decoding step, each beam is expanded using the top-$2k$ tokens to compensate for pruning by the Level 1 filter and preserve diversity. Candidate sequences are ranked by cumulative log-probability, and the top-k candidates are retained after each step.

4 Evaluation

To comprehensively assess Syntropy, we conduct a systematic empirical evaluation addressing the following research questions.
RQ1. How well does Syntropy perform on existing benchmarks, and how does it generalise across different LLMs?

RQ2. How does the performance of Syntropy vary across LLMs fine-tuned with LoRA using training sets of different sizes?

RQ3. How do the Syntropy-Train module, the BNF-style prompt, and the two-level monitoring mechanism in Syntropy-Gen affect overall performance of Syntropy?

RQ4. How robust is Syntropy under different subtyping transformations, including covariance and contravariance (collectively referred to as variance), as well as reordering?

4.1 Experimental Setup

4.1.1 Dataset Construction. The dataset is constructed by collecting *supertypes*, which serve as inputs to the framework. Two sources are considered: *literature-derived data* and *synthetic data*.

The literature-derived data (i.e. existing data) are collected from prior work on Multiparty Session Types (MPST) [5–7, 15], covering real-world communication protocols in domains such as financial transactions, security, distributed coordination, and service orchestration. These protocols provide realistic and semantically meaningful specifications.

To complement these examples and improve structural coverage, synthetic supertypes are additionally constructed to capture a broader range of communication patterns and structures not present in existing benchmarks, enabling evaluation across diverse protocols with varying levels of complexity.

Subtype Generation and Validation. To construct supervised training data, pairs of the form (supertype, subtype) are created, defining the learning task. For synthetic supertypes, subtypes are generated through a heuristic procedure inspired by an asynchronous subtyping algorithm [6] and validated using its checker.

For literature-derived supertypes, no canonical target subtype is available. Benchmark cases contain a limited number of reference subtypes, typically one per supertype, providing only a partial view of the refinement space. To enrich the training data, additional subtypes are generated using the same procedure and validated accordingly. Validation results show that 99.13% of subtypes from literature-derived supertypes (excluding original benchmark subtypes) and 97.97% from synthetic supertypes are accepted, indicating high reliability.

Notably, the asynchronous subtyping algorithm in [6] is sound but *not* complete, and the checker inherits this limitation. Consequently, rejected subtypes may be deemed semantically valid. In addition, practical constraints prevent the checker from handling protocols beyond a certain size threshold (e.g. 701 grammar units), such that even identity subtypes may be rejected.

Dataset Split. Supertypes are partitioned into training and test sets. The training set comprises 704 supertypes, yielding 10,800 (supertype, subtype) pairs (100 from literature-derived data and 604 from synthetic data). To balance learning signals, subtypes generated by different subtyping rules are approximately balanced. The test set consists of 100 representative supertypes (43 from literature-derived data and 57 from synthetic data) and is used as input for evaluation.

4.1.2 Foundation Model Selection. Qwen2.5-Coder-7B-Instruct [25] serves as the primary foundation model for our experiments. Unlike a raw pre-trained base model, it is instruction-tuned and optimised for code and structured generation, making it well suited for grammar-constrained protocol specifications such as MPST session types. The model balances generation quality and computational efficiency, enabling systematic fine-tuning and controlled ablation studies within reasonable resource constraints. For reproducibility, a fixed random seed (seed = 42) is used across all experiments.

To the best of our knowledge, no prior work has explored the use of large language models for type-based generation in the context of MPST. Existing research primarily focuses on formal methods rather than learning-based approaches. As a result, no directly comparable baseline method is available.

4.1.3 Evaluation Metrics. To comprehensively assess the performance of Syntropy, a multi-metric evaluation framework is adopted, targeting two main objectives: *validity* and *diversity*. Validity is evaluated at both syntactic and semantic levels, while diversity captures variations induced by subtyping transformations, including interaction reordering and variance. Semantic validity is further analysed by transformation type. Five metrics are applied:

(1) **Syntactic Validity.** The proportion of generated subtypes that conform to the formal grammar specification, as verified by a dedicated syntax checker.
(2) **Semantic Validity.** The proportion of syntactically valid subtypes accepted by the subtyping checker in [6], used as a proxy for semantic correctness. Since the checker is sound but not complete, this metric provides a lower bound.
(3) **Reordering vs. Variance Validity.** The acceptance rates of reordering-based and variance-based transformations under the subtyping checker.
(4) **Transformation Rule Distribution.** The frequency of subtyping transformation rules applied during subtype generation, including Identity, RefA, RefB, RefIn, RefOut, and Unfold (see §3). A subtype with multiple transformations contributes to each corresponding rule count.
(5) **Output Complexity.** The average numbers of branches and message exchanges per subtype, reflecting the structural diversity of the generated outputs.

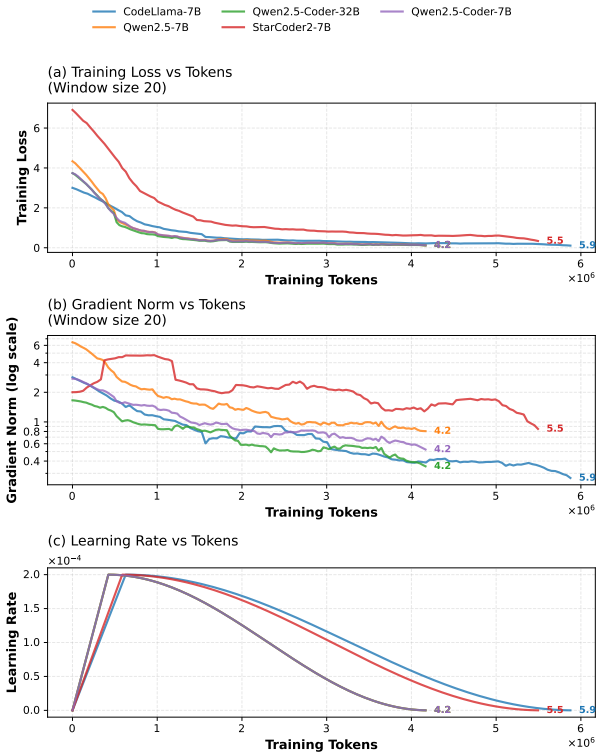


Figure 5: Training dynamics across LLMs

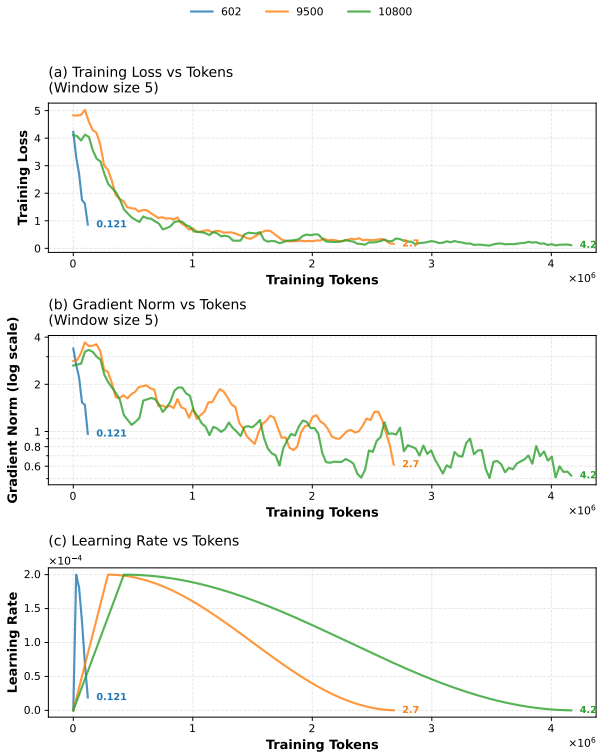


Figure 6: Training dynamics across data scales

Table 2: Training cost across LLMs

Model	Time (min)	Tok (M)
Qwen2.5-Coder-7B	20.33	4.2
CodeLlama-7B	41.19	5.9
StarCoder2-7B	32.51	5.5
Qwen2.5-7B	32.55	4.2
Qwen2.5-Coder-32B	61.61	4.2

In addition, the comparison with frontier language models (§4.7), employs the ***coverage*** metric, defined as the proportion of supertypes for which at least one subtype is generated. This metric measures a model's applicability across diverse protocol specifications.

4.1.4 Training Configuration. Alongside Qwen2.5-Coder-7B-Instruct, we fine-tune four additional open-source language models, CodeLlama-7B-Instruct [43], Qwen2.5-7B-Instruct [42], StarCoder2-7B [29], and Qwen2.5-Coder-32B-Instruct [25], using the same Low-Rank Adaptation (LoRA) configuration with a rank of 64, a learning rate of 2×10^{-4}, and a batch size of 1. All models are instruction-tuned, except for StarCoder2-7B, for which no Instruct variant is publicly available. For brevity, the "-Instruct" suffix is omitted throughout the remainder of the paper, including figures and tables.

Training is conducted for 3 epochs, with costs reported in Table 2. For Qwen2.5-Coder-7B, the average sequence length is 2,027.5 tokens per sample and the training throughput is approximately 3,415 tokens per second.

Fig. 5 illustrates the training loss curves for all models, with steadily decreasing loss indicating stable optimisation dynamics. The gradient optimisation process remains stable, and the learning rate is well tuned, exhibiting near-optimal behaviour. These results suggest that Syntropy can be efficiently fine-tuned with moderate computational cost.

4.2 RQ1. Benchmark Performance and Cross-LLM Generalisation

We evaluate Syntropy on our primary foundation model, Qwen2.5-Coder-7B, using metrics defined in §4.1.3, including syntactic and semantic validity, transformation rule distribution, and output complexity. We further assess its generalisation across additional LLMs, analysing performance consistency and cross-model applicability. Fig. 7 presents the transformation rule distribution, while the remaining metrics and computational cost are summarised in Table 3.

4.2.1 Results on the Primary Foundation Model. We compare two approaches: *direct generation* and *constrained generation* (with Two-Level Monitoring), introduced in §3.2, on the full test set.

Direct Generation (w/o Two-Level Monitoring). Syntactic validity reaches 85.0%, while semantic validity – evaluated on syntactically valid outputs – is 60.4%, reflecting substantial semantic errors despite high syntactic correctness. The transformation rule distribution indicates diverse application of subtyping rules, while each supertype yields approximately 5 subtypes per rule, with an average of 2 branching points and 8 message exchanges.

Generation with Two-Level Monitoring. Syntactic validity increases to 96.7%, while semantic validity among syntactically valid outputs reaches 99.2%, indicating near-perfect semantic correctness.

Table 3: Performance and computational cost across LLMs

Model	No Monitoring				With Monitoring			
	Syn. (%)	**Sem.** (%)	**Br.** / **Msg.**	**Time** (min) / **Tok** (M)	**Syn.** (%)	**Sem.** (%)	**Br.** / **Msg.**	**Time** (min) / **Tok** (M)
Qwen2.5-Coder-7B	85.0	60.4	1.83 / 8.07	335.69 / 8.6	96.7	99.2	1.93 / 8.54	1041.56 / 27.5
CodeLlama-7B	72.6	62.1	1.87 / 7.18	520.15 / 10.4	98.1	99.5	2.22 / 7.00	1733.71 / 33.5
StarCoder2-7B	85.8	48.8	2.04 / 6.67	733.60 / 9.1	95.4	95.7	2.06 / 6.58	1212.12 / 30.2
Qwen2.5-7B	80.4	60.9	1.90 / 7.32	585.19 / 8.6	97.0	96.4	2.01 / 7.22	1606.01 / 27.7
Qwen2.5-Coder-32B	81.6	66.1	1.92 / 6.66	1940.00 / 8.5	96.3	95.6	1.99 / 7.23	3867.99 / 27.4

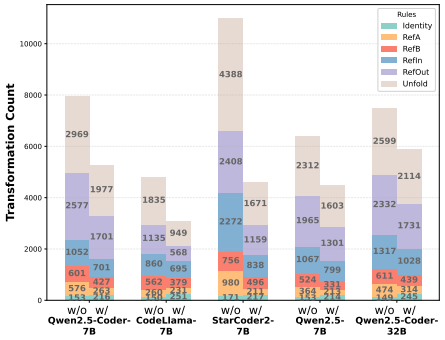
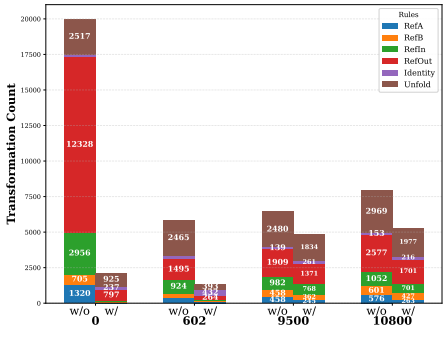
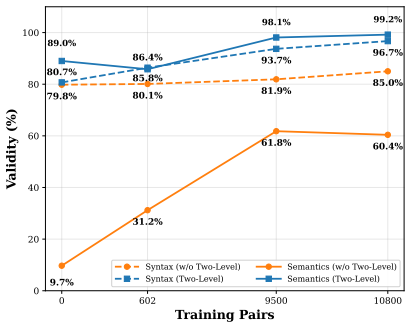


Figure 7: Transformation distribution across LLMs



(a) Transformation distribution across data scales



(b) Validity across data scales

Figure 8: Transformation distribution and validity across data scales

This improvement from 60.4% to 99.2% demonstrates the effectiveness of the monitoring-based approach, while the transformation rule distribution and output complexity remain comparable.

Overhead Analysis. We analyse the computational overhead introduced by Two-Level Monitoring. Enabling monitoring increases token consumption from 8.6M to 27.5M (approximately 3×), mainly due to additional validation and constraint enforcement. While runtime is reported in Table 3, it is not used as the primary cost metric due to its sensitivity to GPU allocation variability in the HTC cluster; token-level overhead serves as the basis for subsequent analysis.

Despite this additional cost, the monitoring mechanism substantially improves semantic validity, indicating a clear trade-off between efficiency and correctness. Direct generation is more suitable for efficiency-oriented scenarios, whereas monitoring-based generation is preferable for correctness-critical applications.

4.2.2 Cross-LLM Generalisation. We evaluate Syntropy across multiple LLMs on the full test set, including CodeLlama-7B, StarCoder2-7B, Qwen2.5-7B, and Qwen2.5-Coder-32B. As shown in Table 3, syntactic validity remains consistently high (95.4%–98.1%), while semantic validity reaches up to 99.5%, demonstrating robust generalisation across models. These results indicate that Syntropy is not tied to a specific architecture and exhibits strong cross-model applicability.

The transformation rule distribution (Fig. 7) is generally consistent, with some variations among models. In particular, CodeLlama-7B exhibits limited diversity under default decoding settings, failing to generate certain transformations (e.g. RefA and RefB). To address this, we adopt stochastic beam search with temperature scaling ($T = 1.2$), which restores transformation diversity without modifying the framework. StarCoder2-7B shows a more imbalanced distribution compared to other models, which is qualitatively consistent with the slower convergence reflected in the training dynamics (Fig. 5).

Table 4: Complexity and computational cost across data scales

Pairs	No Monitoring		With Monitoring	
	Br. / **Msg.**	**Time** (min) / **Tok** (M)	**Br.** / **Msg.**	**Time** (min) / **Tok** (M)
0	1.76 / 6.78	1578.16 / 8.5	2.03 / 8.06	580.65 / 27.3
602	1.95 / 5.86	338.48 / 8.5	2.35 / 6.73	1120.37 / 27.1
9500	2.01 / 8.12	486.02 / 8.5	2.13 / 7.66	1575.50 / 27.3
10800	1.83 / 8.06	335.69 / 8.6	1.93 / 8.54	1041.56 / 27.5

Model-level overhead remains consistent, exhibiting trends similar to those observed for the primary foundation model. This suggests that the computational characteristics of Syntropy scale predictably across different LLM architectures.

Answer to RQ1. Syntropy achieves strong performance on benchmark datasets and generalises effectively across different LLMs. Two-Level Monitoring significantly improves semantic validity while maintaining comparable generation behaviour and predictable computational overhead.

4.3 RQ2. Impact of Training Data Scale on Performance

To study how training data scale affects performance, we fine-tune models with LoRA on increasing numbers of (supertype, subtype) pairs. Due to variability in subtype generation, we approximate three regimes (small, large, and near-saturation) using 602, 9, 500, and 10, 800 pairs, respectively, along with a no-fine-tuning baseline.

Fig. 6 reveals that training data size significantly affects convergence behaviour, while transformation rule distributions and both syntactic and semantic validity vary accordingly, as shown in

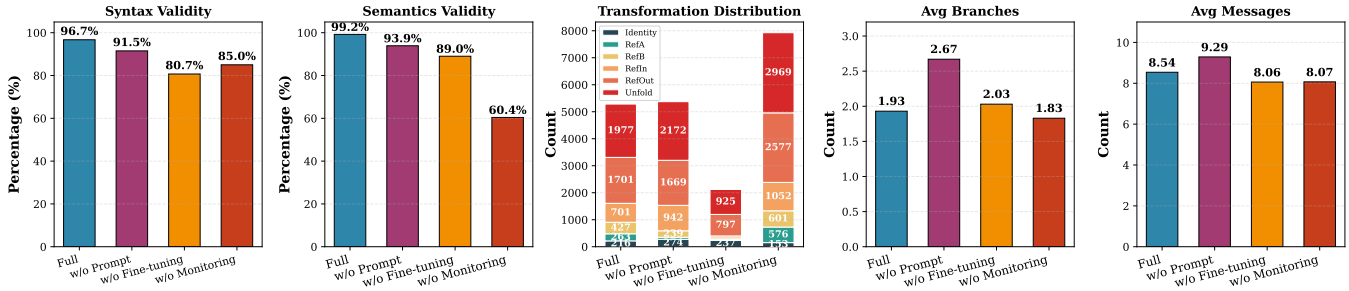


Figure 9: Ablation results across metrics on Qwen2.5-Coder-7B

Fig. 8a and Fig. 8b. Additional metrics and computational costs are reported in Table 4.

No fine-tuning (0 pairs). Without fine-tuning, semantic validity is low (9.7%), and generated subtypes are structurally simple, with limited transformation diversity and few reordering cases. Two-Level Monitoring improves validity substantially but does not address the lack of structural diversity, highlighting the need for fine-tuning to generate valid and diverse subtypes.

Small-scale (602 pairs). Fine-tuning yields significant performance gains. Pre-monitoring semantic validity increases from 9.7% to 31.2%, while Two-Level Monitoring maintains validity at 85–90%. Structural diversity also improves, with more frequent complex transformations. These results indicate that even limited training data provides meaningful improvements, although performance remains far from saturation.

Large-scale (9,500 pairs). At this scale, performance reaches a near-optimal level. Pre-monitoring semantic validity approximately doubles relative to the 602-pair setting, reaching 98% after Two-Level Monitoring. Structural diversity is well balanced, with broader coverage of complex transformations. This is consistent with the model having largely converged, with limited scope for further improvement.

Near-saturation (10,800 pairs). Increasing the training size from 9,500 to 10,800 pairs yields only marginal changes (approximately 1%), indicating that performance has effectively saturated. This implies diminishing returns from further scaling and confirms that the model has reached a near-saturation regime.

Computational Cost. As shown in Table 4, computational cost exhibits minimal variation as the training dataset scales from 602 to 10,800 pairs, incurring only minor differences in token consumption. This shows that data scaling has limited impact on efficiency. The relative overhead introduced by Two-Level Monitoring remains consistent with the analysis in § 4.2.1. Overall, the framework demonstrates stable and predictable computational behaviour across different scales.

Answer to RQ2. Performance improves rapidly as training data increases, but saturates around 9,500 pairs, beyond which additional data yields only marginal gains.

4.4 RQ3. Ablation Study of Syntropy Components

To understand the contribution of each component in Syntropy, we conduct ablation studies on Qwen2.5-Coder-7B using three variants: *w/o Prompt*, *w/o Fine-tuning*, and *w/o Two-Level Monitoring*. As the impact of fine-tuning has been analysed in RQ2 (§ 4.3), we focus on prompting and monitoring. The results of the ablation study on validity, transformation rule distribution, and output complexity are illustrated in Fig. 9.

w/o Two-Level Monitoring. This configuration exhibits the most severe degradation. Semantic validity drops to 60.4%, considerably lower than all other configurations (≥ 85%), highlighting that Two-Level Monitoring is essential for ensuring correctness.

w/o Prompt. Removing the BNF-style prompt does not substantially degrade validity, as both syntactic and semantic validity remain above 90%. However, semantic validity decreases from 99.2% to 93.9%, indicating that prompting contributes to achieving near-perfect correctness. In contrast, structural diversity decreases more noticeably, with fewer reordering transformations, highlighting the importance of prompting for balanced transformation coverage.

Full Syntropy framework. The complete system achieves the best overall performance, with 99.2% semantic validity and balanced structural diversity.

Answer to RQ3. The components serve complementary roles: prompting enhances structural diversity, and Two-Level Monitoring ensures correctness. Combined with fine-tuning (RQ2), the full framework achieves the best overall performance. Removing any component degrades performance, with monitoring being most critical for validity and prompting for diversity.

4.5 RQ4. Robustness under Reordering and Variance Transformations

To evaluate the robustness of Syntropy under different subtyping transformations, we analyse performance across two core categories: reordering (RefA and RefB) and variance (RefIn and RefOut). As shown in Table 5, both achieve consistently high semantic validity across all evaluated models (89.8%–100%), demonstrating strong robustness even for structurally complex reordering.

However, the generated subtype distribution is not balanced, with variance cases consistently outnumbering reordering ones. To investigate this, we introduce two representative structural cases:

Table 5: Semantic validity of diverse transformations across LLMs

	Reordering			Variance		
Model	**RefA**	**RefB**	**Avg.**	**RefIn**	**RefOut**	**Avg.**
Qwen2.5-Coder-7B	261/263 (99.2%)	421/427 (98.6%)	673/681 (98.8%)	681/701 (97.1%)	1649/1701 (96.9%)	2167/2234 (97.0%)
CodeLlama-7B	230/231 (99.6%)	377/379 (99.5%)	606/609 (99.5%)	683/695 (98.3%)	554/568 (97.5%)	1100/1125 (97.8%)
StarCoder2-7B	210/211 (99.5%)	485/496 (97.8%)	682/694 (98.3%)	773/838 (92.2%)	1010/1159 (87.1%)	1695/1887 (89.8%)
Qwen2.5-7B	213/213 (100%)	331/331 (100%)	544/544(100%)	739/799 (92.5%)	1209/1301 (92.9%)	1789/1939 (92.3%)
Qwen2.5-Coder-32B	314/314 (100.0%)	433/439 (98.6%)	744/750 (99.2%)	973/1028 (94.6%)	1612/1731 (93.1%)	2378/2545 (93.4%)

Table 6: Transformation preferences across two structural cases on Qwen2.5-Coder-7B

	Reordering			Variance		
Case	**RefA**	**RefB**	**Avg.**	**RefIn**	**RefOut**	**Avg.**
T_1	14/15 (93.3%)	4/4 (100%)	18/19 (94.7%)	13/13 (100%)	32/33 (97.0%)	39/40 (97.5%)
$T_1^{\hookrightarrow}$	19/19 (100%)	8/8 (100%)	27/27 (100%)	16/16 (100%)	35/35 (100%)	46/46 (100%)
T_2	0	14/14 (100%)	14/14 (100%)	12/12 (100%)	38/38 (100%)	41/41(100%)

T_1 = REC_X_OPEN p?upd$_p$; q?upd$_q$; lbrace m!std; REC_X_CLOSE, m!wtd; REC_X_CLOSE rbrace

T_2 = REC_X_OPEN q?upd$_q$; lbrace m!std; REC_X_CLOSE, m!wtd; REC_X_CLOSE rbrace

and generate their subtypes to analyse the resulting transformation distribution on Qwen2.5-Coder-7B. In addition, we consider a *strengthening reordering* setting applied to T_1, where constraints are introduced to encourage reordering. The results are summarised in Table 6, where $T_1^{\hookrightarrow}$ denotes increased reordering.

For T_1, both RefA and RefB are allowed, yet the generated subtypes favour RefA, indicating a bias toward transformations aligned with the prefix structure. For T_2, removing the initial receive action enables RefB-style reordering; consequently, RefA disappears while RefB increases, confirming that both reordering types are supported when structurally enabled.

Across both cases, variance transformations exceed reordering by approximately two to three times, suggesting that they are easier to produce. While strengthening reordering constraints increases their proportion, variance remains dominant, indicating a persistent preference for simpler patterns.

Answer to RQ4. Syntropy maintains high semantic validity across both reordering and variance transformations, demonstrating strong robustness. However, the model exhibits a preference for simpler (variance) transformations, while reordering can be partially steered through constraints.

4.6 Error Analysis and Threats to Validity

Syntax Validity. LLMs occasionally produce syntactic errors, including mismatched parentheses, incomplete expressions (e.g., ending with < or ;), and unbound recursive variables. These errors reduce syntactic validity and are not fully eliminated by syntax normalisation during fine-tuning. This suggests that LLMs may be less reliable when handling mathematically structured formats compared to more common code-like syntax. Rather than introducing dedicated syntax correction techniques, we mitigate these errors indirectly: the two-level monitoring framework filters many invalid outputs, while multiple generations ensure a sufficient number of syntactically valid subtypes.

Semantic Validity. Ensuring semantic validity is challenging due to the undecidable nature of subtyping and the incompleteness of the subtyping checker used for validation [6]. While accepted subtypes are guaranteed to be correct, rejected cases cannot be conclusively deemed incorrect, introducing uncertainty in evaluating borderline cases.

Threats to Validity. The validity and diversity of generated subtypes depend on the characteristics of the underlying LLM. Although fine-tuning improves performance, generation quality remains influenced by the choice of model and may vary under different configurations.

In addition, our evaluation focuses exclusively on subtype generation. While consistent performance is observed across multiple LLMs, we do not assess generalisation to other generation tasks. Extending the approach beyond subtyping remains an important direction for future work.

4.7 Comparison with Frontier Language Models

To assess the necessity of a task-specific framework in the presence of modern frontier language models, we compare Syntropy with GPT-5.5 [37] and DeepSeek-V4-Pro [16], representative leading proprietary and open-weight models, respectively, in terms of both effectiveness and computational cost.

Both models are evaluated on the benchmark using the *same* prompts as the fine-tuned models under their default inference configurations. Table 7 summarises the results, including ***coverage***.

Performance Analysis. The key difference between frontier and fine-tuned models lies in their ability to generate subtypes across the benchmark. As shown in Table 7, GPT-5.5 and DeepSeek-V4-Pro cover 18% and 4% of benchmark supertypes, respectively. In contrast, all five fine-tuned models provide **full** coverage.

Among the generated outputs, GPT-5.5 and DeepSeek-V4-Pro exhibit syntactic and semantic validity broadly consistent with those of the fine-tuned models (Table 3). GPT-5.5 achieves 94.59% syntactic validity and 99.29% semantic validity, while DeepSeek-V4-Pro records 100.00% and 95.65%, respectively. GPT-5.5 additionally produces subtypes spanning all transformation categories and shows comparable branch and message counts. However, these metrics need to be interpreted in the context of their substantially lower coverage. We further observe an overall tendency for covered supertypes to involve fewer message exchanges and simpler structures.

Cost Analysis. Based on the results from Tables 2, 3 and 7, prompting frontier models requires fewer tokens and less time than the combined cost of fine-tuning and evaluating the models used in this work. For the latter, the figures include both model training (Table 2) and benchmark generation (Table 3), with training accounting for a modest fraction of the overall expenditure. Following §4.2, token consumption remains the primary basis for comparison. The lower token usage of frontier models is partly attributable to their limited

Table 7: Performance and computational cost of prompting frontier models

	Supertype-Level	Generated Subtypes									Cost	
Model	**Cov.** (%)	**Syn.** (%)	**Sem.** (%)	**Br.** / **Msg.**	RefA	RefB	RefIn	RefOut	Unfold	Identity	**Time** (min)	**Tok** (M)
GPT-5.5	18	94.59	99.29	1.78 / 7.30	18	2	49	13	50	52	59.47	0.5
DeepSeek-V4-Pro	4	100.00	95.65	1.00 / 5.35	4	0	11	0	0	16	41.47	0.5

coverage and significantly fewer subtype candidates per covered supertype, averaging 8.22 and 5.75 for GPT-5.5 and DeepSeek-V4-Pro, respectively, compared with 19.20–35.30 for the fine-tuned models.

Discussion. Taken together, these findings indicate that, although frontier models can generate valid subtype candidates at lower computational cost, they do not provide sufficient coverage for comprehensive subtype generation. The effectiveness gains achieved by the fine-tuned Syntropy models, while maintaining high validity, therefore justify the need for this work.

5 Related Work

Asynchronous Subtyping. Asynchronous subtyping was initially shown to be sound and complete for binary session types (i.e. involving two participants) in [11]. It was subsequently extended to multiparty session types [22], with its formalisation and correctness proof mechanised in Rocq [18]. For a comprehensive survey of asynchronous subtyping, see [10].

Due to the undecidability of asynchronous subtyping [8, 27], even in the binary case, existing work focuses on developing sound, though necessarily incomplete, algorithms for subtyping verification. Approaches such as [5, 7] provide practical checking procedures for binary session types, while [6, 15] extend these ideas to multiparty settings based on a formal definition of asynchronous multiparty subtyping [22], which precisely characterises safety-preserving type refinements, thereby guaranteeing correctness, including deadlock freedom. Benchmarks from [5–7, 15] are adopted in the training and evaluation datasets of our framework, while the approach and checker in [6] serve as the basis for our constrained generation and semantic validity checking.

However, the automatic generation of valid asynchronous subtypes remains unexplored in existing work.

LLMs for Formal Specification. Transformer-based language models [48], such as Qwen-Coder [25], CodeLlama [43], and StarCoder [29], have substantially advanced natural language tasks such as summarisation [13], code generation [9], and program synthesis [2]. Leveraging these strengths, a line of work [12, 14, 19, 31, 34, 45, 53, 54] investigates the translation of natural language into formal specifications, particularly temporal logic, to reduce the effort and error-proneness. Beyond specification generation, recent studies explore improving the reliability of LLM-based program synthesis by integrating formal reasoning techniques [30, 50], including theorem proving, static analysis, and deductive verification.

These efforts demonstrate the complementary strengths of LLMs and formal methods in supporting the development of trustworthy software systems, motivating our exploration of integrating LLMs with the formal specification and verification of communication protocols.

Constrained Generation with LLMs. Constrained generation aims to ensure that LLM outputs satisfy specified constraints and has mainly been studied in the context of constrained decoding. Syntactic constraints, particularly grammar-constrained decoding based on context-free grammars, have been extensively explored [3, 4, 20, 38, 40, 47, 49, 51]. Simple context-sensitive features, such as indentation in Python and scope markers in Go, have also been incorporated [33, 47].

In addition to syntactic constraints, recent work has investigated the enforcement of semantic constraints, such as type safety [35, 36], as well as more general mechanisms, including monitors [1], which enable user-defined constraint checking during decoding. These approaches inspire the design of our two-level monitoring strategy within constrained generation to enforce semantic correctness constraints of generated protocol refinements in Syntropy.

LLM-based Communication Protocols. The automatic generation of communication protocols among LLM agents has been explored, where emergent communication frameworks show that shared protocols can arise from decentralised interactions [46], while alternative schemes, such as embedding-based protocols, enable richer information exchange [39]. In addition, LLMs have been used to dynamically construct or adapt communication protocols at runtime [32, 41]. These approaches shift from static, human-designed protocols toward adaptive and learned communication paradigms; however, their underlying motivation differs from ours and is not grounded in formal specifications.

6 Conclusion

In this paper, we explored how large language models can be extended beyond syntactic code generation to provide behavioural guarantees in communication protocols. By embedding formal constraints into the LLM-based generation process, the proposed framework Syntropy enables the construction of protocol refinements that preserve properties such as deadlock freedom. The results demonstrate that integrating generative models with formal specification and verification techniques facilitates correctness-preserving synthesis and highlights promising directions for applying LLMs to safety-critical software engineering tasks.

In future work, we intend to extend the approach to more general formalisms, including choreographies, contract-based models, and temporal logics. We further plan to investigate verifier-guided iterative refinement techniques and the integration of Syntropy into practical development workflows, including protocol evolution, interactive design assistance, and verification-aware code generation within existing toolchains.

Data Availability Statement

The artifact supporting this work, including the implementation of Syntropy, trained models, evaluation data, and evaluations of frontier language models, is publicly available via DOI: https://doi.org/10.5281/zenodo.21737877. The repository contains all source code, datasets, and instructions required to reproduce the experiments and results reported in this paper.

References

[1] Lakshya A. Agrawal, Aditya Kanade, Navin Goyal, Shuvendu K. Lahiri, and Sriram K. Rajamani. 2023. Monitor-Guided Decoding of Code LMs with Static Analysis of Repository Context. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/662b1774ba8845fc1fa3d1fc0177ceeb-Abstract-Conference.html

[2] Jacob Austin, Augustus Odena, Maxwell I. Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie J. Cai, Michael Terry, Quoc V. Le, and Charles Sutton. 2021. Program Synthesis with Large Language Models. *CoRR* abs/2108.07732 (2021). arXiv:2108.07732 https://arxiv.org/abs/2108.07732

[3] Luca Beurer-Kellner, Marc Fischer, and Martin T. Vechev. 2023. Prompting Is Programming: A Query Language for Large Language Models. *Proc. ACM Program. Lang.* 7, PLDI (2023), 1946–1969. doi:10.1145/3591300

[4] Luca Beurer-Kellner, Marc Fischer, and Martin T. Vechev. 2024. Guiding LLMs The Right Way: Fast, Non-Invasive Constrained Generation. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024 (Proceedings of Machine Learning Research, Vol. 235)*, Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (Eds.). PMLR / OpenReview.net, 3658–3673. https://proceedings.mlr.press/v235/beurer-kellner24a.html

[5] Laura Bocchi, Andy King, and Maurizio Murgia. 2024. Asynchronous Subtyping by Trace Relaxation. In *Tools and Algorithms for the Construction and Analysis of Systems - 30th International Conference, TACAS 2024, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2024, Luxembourg City, Luxembourg, April 6-11, 2024, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 14570)*, Bernd Finkbeiner and Laura Kovács (Eds.). Springer, 207–226. doi:10.1007/978-3-031-57246-3_12

[6] Laura Bocchi, Andy King, Maurizio Murgia, and Simon Thompson. 2025. Abstract Subtyping for Asynchronous Multiparty Sessions. In *36th International Conference on Concurrency Theory, CONCUR 2025, Aarhus, Denmark, August 26-29, 2025 (LIPIcs, Vol. 348)*, Patricia Bouyer and Jaco van de Pol (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 10:1–10:19. doi:10.4230/LIPICS.CONCUR.2025.10

[7] Mario Bravetti, Marco Carbone, Julien Lange, Nobuko Yoshida, and Gianluigi Zavattaro. 2019. A Sound Algorithm for Asynchronous Session Subtyping. In *30th International Conference on Concurrency Theory, CONCUR 2019, Amsterdam, The Netherlands, August 27-30, 2019 (LIPIcs, Vol. 140)*, Wan J. Fokkink and Rob van Glabbeek (Eds.). Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 38:1–38:16. doi:10.4230/LIPICS.CONCUR.2019.38

[8] Mario Bravetti, Marco Carbone, and Gianluigi Zavattaro. 2017. Undecidability of asynchronous session subtyping. *Inf. Comput.* 256 (2017), 300–320. doi:10.1016/J.IC.2017.07.010

[9] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Pondé de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Joshua Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *CoRR* abs/2107.03374 (2021). arXiv:2107.03374 https://arxiv.org/abs/2107.03374

[10] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, and Nobuko Yoshida. 2024. On the Preciseness of Subtyping in Session Types: 10 Years Later. In *Proceedings of the 26th International Symposium on Principles and Practice of Declarative Programming, PPDP 2024, Milano, Italy, September 9-11, 2024*, Alessandro Bruni, Alberto Momigliano, Matteo Pradella, Matteo Rossi, and James Cheney (Eds.). ACM, 2:1–2:3. doi:10.1145/3678232.3678258

[11] Tzu-Chun Chen, Mariangiola Dezani-Ciancaglini, Alceste Scalas, and Nobuko Yoshida. 2017. On the Preciseness of Subtyping in Session Types. *Logical Methods in Computer Science* 13, 2 (2017). doi:10.23638/LMCS-13(2:12)2017

[12] Yongchao Chen, Rujul Gandhi, Yang Zhang, and Chuchu Fan. 2023. NL2TL: Transforming Natural Languages to Temporal Logics using Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, Singapore, 15880–15903. doi:10.18653/v1/2023.emnlp-main.985

[13] Bharath Chintagunta, Namit Katariya, Xavier Amatriain, and Anitha Kannan. 2021. Medically Aware GPT-3 as a Data Generator for Medical Dialogue Summarization. In *Proceedings of the 6th Machine Learning for Healthcare Conference (Proceedings of Machine Learning Research, Vol. 149)*, Ken Jung, Serena Yeung, Mark Sendak, Michael Sjoding, and Rajesh Ranganath (Eds.). PMLR, 354–372. https://proceedings.mlr.press/v149/chintagunta21a.html

[14] Matthias Cosler, Christopher Hahn, Daniel Mendoza, Frederik Schmitt, and Caroline Trippel. 2023. nl2spec: Interactively Translating Unstructured Natural Language to Temporal Logics with Large Language Models. In *Computer Aided Verification - 35th International Conference, CAV 2023, Paris, France, July 17-22, 2023, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 13965)*, Constantin Enea and Akash Lal (Eds.). Springer, 383–396. doi:10.1007/978-3-031-37703-7_18

[15] Zak Cutner, Nobuko Yoshida, and Martin Vassor. 2022. Deadlock-free asynchronous message reordering in rust with multiparty session types. In *PPoPP '22: 27th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Seoul, Republic of Korea, April 2 - 6, 2022*, Jaejin Lee, Kunal Agrawal, and Michael F. Spear (Eds.). ACM, 246–261. doi:10.1145/3503221.3508404

[16] DeepSeek-AI. 2026. DeepSeek-V4: Towards Highly Efficient Million-Token Context Intelligence. arXiv:2606.19348 https://arxiv.org/abs/2606.19348

[17] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch-Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. 2017. Microservices: Yesterday, Today, and Tomorrow. In *Present and Ulterior Software Engineering*, Manuel Mazzara and Bertrand Meyer (Eds.). Springer, 195–216. doi:10.1007/978-3-319-67425-4_12

[18] Burak Ekici and Nobuko Yoshida. 2026. Formalising Asynchronous Session Subtyping. *ACM Trans. Comput. Logic* 27, 3, Article 18 (June 2026), 45 pages. doi:10.1145/3815176

[19] Francesco Fuggitti and Tathagata Chakraborti. 2023. NL2LTL - a Python Package for Converting Natural Language (NL) Instructions to Linear Temporal Logic (LTL) Formulas. In *Thirty-Seventh AAAI Conference on Artificial Intelligence, AAAI 2023, Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence, IAAI 2023, Thirteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2023, Washington, DC, USA, February 7-14, 2023*, Brian Williams, Yiling Chen, and Jennifer Neville (Eds.). AAAI Press, 16428–16430. doi:10.1609/AAAI.V37I13.27068

[20] Saibo Geng, Martin Josifoski, Maxime Peyrard, and Robert West. 2023. Grammar-Constrained Decoding for Structured NLP Tasks without Finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 10932–10952. doi:10.18653/V1/2023.EMNLP-MAIN.674

[21] Silvia Ghilezan, Svetlana Jaksic, Jovanka Pantovic, Alceste Scalas, and Nobuko Yoshida. 2019. Precise subtyping for synchronous multiparty sessions. *J. Log. Algebraic Methods Program.* 104 (2019), 127–173. doi:10.1016/J.JLAMP.2018.12.002

[22] Silvia Ghilezan, Jovanka Pantović, Ivan Prokić, Alceste Scalas, and Nobuko Yoshida. 2023. Precise Subtyping for Asynchronous Multiparty Sessions. *ACM Transactions on Computational Logic* Volume 24, Issue 2, 14 (2023), 1–73. doi:10.1145/3568422

[23] Kohei Honda, Vasco Thudichum Vasconcelos, and Makoto Kubo. 1998. Language Primitives and Type Discipline for Structured Communication-Based Programming. In *ESOP*. doi:10.1007/BFb0053567

[24] Kohei Honda, Nobuko Yoshida, and Marco Carbone. 2016. Multiparty Asynchronous Session Types. *J. ACM* 63, 1, Article 9 (2016). doi:10.1145/2827695

[25] Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, Kai Dang, Yang Fan, Yichang Zhang, An Yang, Rui Men, Fei Huang, Bo Zheng, Yibo Miao, Shanghaoran Quan, Yunlong Feng, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-Coder Technical Report. arXiv:2409.12186 [cs.CL] https://arxiv.org/abs/2409.12186

[26] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh

Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2021. Advances and Open Problems in Federated Learning. *Found. Trends Mach. Learn.* 14, 1-2 (2021), 1–210. doi:10.1561/2200000083

[27] Julien Lange and Nobuko Yoshida. 2017. On the Undecidability of Asynchronous Session Subtyping. In *Foundations of Software Science and Computation Structures - 20th International Conference, FOSSACS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings (Lecture Notes in Computer Science, Vol. 10203)*, Javier Esparza and Andrzej S. Murawski (Eds.). 441–457. doi:10.1007/978-3-662-54458-7_26

[28] Edward A. Lee. 2008. Cyber Physical Systems: Design Challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*. 363–369. doi:10.1109/ISORC.2008.25

[29] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy V, Jason T. Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! *Trans. Mach. Learn. Res.* 2023 (2023). https://openreview.net/forum?id=KoFOg41haE

[30] Xiaohan Lin, Qingxing Cao, Yinya Huang, Haiming Wang, Jianqiao Lu, Zhengying Liu, Linqi Song, and Xiaodan Liang. 2024. FVEL: Interactive Formal Verification Environment with Large Language Models via Theorem Proving. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*, Amir Globersons, Lester Mackey, Danielle Belgrave, Angela Fan, Ulrich Paquet, Jakub M. Tomczak, and Cheng Zhang (Eds.). http://papers.nips.cc/paper_files/paper/2024/hash/62c6d7893b13a13c659cb815852dd00d-Abstract-Datasets_and_Benchmarks_Track.html

[31] Zhi Ma, Cheng Wen, Zhexin Su, Xiao Liang, Cong Tian, Shengchao Qin, and Mengfei Yang. 2025. Bridging Natural Language and Formal Specification-Automated Translation of Software Requirements to LTL via Hierarchical Semantics Decomposition Using LLMs. In *40th IEEE/ACM International Conference on Automated Software Engineering, ASE 2025, Seoul, Korea, Republic of, November 16-20, 2025*. IEEE, 1208–1220. doi:10.1109/ASE63991.2025.00104

[32] Samuele Marro, Emanuele La Malfa, Jesse Wright, Guohao Li, Nigel Shadbolt, Michael J. Wooldridge, and Philip Torr. 2024. A Scalable Communication Protocol for Networks of Large Language Models. *CoRR* abs/2410.11905 (2024). arXiv:2410.11905 doi:10.48550/ARXIV.2410.11905

[33] Daniel Melcer, Nathan Fulton, Sanjay Krishna Gouda, and Haifeng Qian. 2024. Constrained Decoding for Fill-in-the-Middle Code Language Models via Efficient Left and Right Quotienting of Context-Sensitive Grammars. arXiv:2402.17988 [cs.PL] https://arxiv.org/abs/2402.17988

[34] Daniel Mendoza, Christopher Hahn, and Caroline Trippel. 2024. Translating Natural Language to Temporal Logics with Large Language Models and Model Checkers. In *Formal Methods in Computer-Aided Design, FMCAD 2024, Prague, Czech Republic, October 15-18, 2024*, Nina Narodytska and Philipp Rümmer (Eds.). IEEE, 1–11. doi:10.34727/2024/ISBN.978-3-85448-065-5_17

[35] Niels Mündler, Jingxuan He, Hao Wang, Koushik Sen, Dawn Song, and Martin T. Vechev. 2025. Type-Constrained Code Generation with Language Models. *Proc. ACM Program. Lang.* 9, PLDI (2025), 601–626. doi:10.1145/3729274

[36] Shaan Nagy, Timothy Zhou, Nadia Polikarpova, and Loris D'Antoni. 2026. ChopChop: A Programmable Framework for Semantically Constraining the Output of Language Models. *Proc. ACM Program. Lang.* 10, POPL (2026), 1905–1932. doi:10.1145/3776708

[37] OpenAI. 2026. Introducing GPT-5.5. https://openai.com/index/introducing-gpt-5-5/. Accessed: June 2026.

[38] Kanghee Park, Timothy Zhou, and Loris D'Antoni. 2025. Flexible and Efficient Grammar-Constrained Decoding. arXiv:2502.05111 [cs.CL] https://arxiv.org/abs/2502.05111

[39] Chau Pham, Boyi Liu, Yingxiang Yang, Zhengyu Chen, Tianyi Liu, Jianbo Yuan, Bryan A. Plummer, Zhaoran Wang, and Hongxia Yang. 2024. Let Models Speak Ciphers: Multiagent Debate through Embeddings. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net. https://openreview.net/forum?id=sehRvaIPQQ

[40] Gabriel Poesia, Alex Polozov, Vu Le, Ashish Tiwari, Gustavo Soares, Christopher Meek, and Sumit Gulwani. 2022. Synchromesh: Reliable Code Generation from Pre-trained Language Models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net. https://openreview.net/forum?id=KmtVD97J43e

[41] Sunil Prakash. 2026. LDP: An Identity-Aware Protocol for Multi-Agent LLM Systems. arXiv:2603.08852 [cs.AI] https://arxiv.org/abs/2603.08852

[42] Qwen, :, An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2025. Qwen2.5 Technical Report. arXiv:2412.15115 [cs.CL] https://arxiv.org/abs/2412.15115

[43] Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton-Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2023. Code Llama: Open Foundation Models for Code. *CoRR* abs/2308.12950 (2023). arXiv:2308.12950 doi:10.48550/ARXIV.2308.12950

[44] Alceste Scalas and Nobuko Yoshida. 2019. Less is More: Multiparty Session Types Revisited. *Proc. ACM Program. Lang.* 3, POPL, Article 30 (Jan. 2019), 29 pages. doi:10.1145/3290343

[45] David Smith Sundarsingh, Jun Wang, Jyotirmoy V. Deshmukh, and Yiannis Kantaros. 2026. ConformalNL2LTL: Translating Natural Language Instructions into Temporal Logic Formulas with Conformal Correctness Guarantees. arXiv:2504.21022 [cs.CL] https://arxiv.org/abs/2504.21022

[46] Tadahiro Taniguchi, Ryo Ueda, Tomoaki Nakamura, Masahiro Suzuki, and Akira Taniguchi. 2025. Generative Emergent Communication: Large Language Model is a Collective World Model. *CoRR* abs/2501.00226 (2025). arXiv:2501.00226 doi:10.48550/ARXIV.2501.00226

[47] Shubham Ugare, Tarun Suresh, Hangoo Kang, Sasa Misailovic, and Gagandeep Singh. 2025. SynCode: LLM Generation with Grammar Augmentation. *Transactions on Machine Learning Research* 2025 (2025). https://openreview.net/forum?id=HiUZtgAPoH

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. 2017. Attention is All you Need. In *Advances in Neural Information Processing Systems*, I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (Eds.), Vol. 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

[49] Bailin Wang, Zi Wang, Xuezhi Wang, Yuan Cao, Rif A. Saurous, and Yoon Kim. 2023. Grammar Prompting for Domain-Specific Language Generation with Large Language Models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.). http://papers.nips.cc/paper_files/paper/2023/hash/cd40d0d65bfebb894ccc9ea822b47fa8-Abstract-Conference.html

[50] Zhongyi Wang, Tengjie Lin, Mingshuai Chen, Mingqi Yang, Haokun Li, Xiao Yi, Shengchao Qin, and Jianwei Yin. 2025. Preguss: It Analyzes, It Specifies, It Verifies. arXiv:2508.14532 [cs.SE] https://arxiv.org/abs/2508.14532

[51] Brandon T. Willard and Rémi Louf. 2023. Efficient Guided Generation for Large Language Models. arXiv:2307.09702 [cs.CL] https://arxiv.org/abs/2307.09702

[52] Nobuko Yoshida. 2024. Programming Language Implementations with Multiparty Session Types. In *Active Object Languages: Current Research Trends*, Frank S. de Boer, Ferruccio Damiani, Reiner Hähnle, Einar Broch Johnsen, and Eduard Kamburjan (Eds.). Lecture Notes in Computer Science, Vol. 14360. Springer, 147–165. doi:10.1007/978-3-031-51060-1_6

[53] Mengyan Zhao, Ran Tao, Yanhong Huang, Jianqi Shi, Shengchao Qin, and Yang Yang. 2024. NL2CTL: Automatic Generation of Formal Requirements Specifications via Large Language Models. In *Formal Methods and Software Engineering: 25th International Conference on Formal Engineering Methods, ICFEM 2024, Hiroshima, Japan, December 2–6, 2024, Proceedings* (Hiroshima, Japan). Springer-Verlag, Berlin, Heidelberg, 1–17. doi:10.1007/978-981-96-0617-7_1

[54] Mengyan Zhao, Ran Tao, Yanhong Huang, Jianqi Shi, Shengchao Qin, and Yang Yang. 2024. NL2CTL: Automatic Generation of Formal Requirements Specifications via Large Language Models. In *Formal Methods and Software Engineering - 25th International Conference on Formal Engineering Methods, ICFEM 2024, Hiroshima, Japan, December 2-6, 2024, Proceedings (Lecture Notes in Computer Science, Vol. 15394)*, Kazuhiro Ogata, Dominique Méry, Meng Sun, and Shaoying Liu (Eds.). Springer, 1–17. doi:10.1007/978-981-96-0617-7_1