Script Choice in LLMs: Evidence for Late-Layer Commitment

David Kletz[†], **Sandra Mitrović**[†], **Itay Sabato**[§], **Ljiljana Dolamić**[‡], **Fabio Rinaldi**[†]
[†] SUPSI, IDSIA, Switzerland
[‡] armasuisse, Science & Technology, Switzerland
[§] Independent Researcher
{david.kletz, sandra.mitrovic, fabio.rinaldi}@supsi.ch
ljiljana.dolamic@armasuisse.ch
itaysabato@gmail.com

Abstract

In this paper, we investigate how script knowledge is distributed across the layers of LLMs using two complementary interpretability methods: logistic regression probing and logit-lens analysis. Our probing experiments reveal a clear asymmetry: both the input script and the instructed output script are encoded in the earliest layers of the network, while, in contrast, commitment to the actual output script emerges only in the final layers, with the model's intermediate representations defaulting to Latin throughout most of the layers. This two-stage process is confirmed by logit-lens analyses, which show that script commitment consistently occurs at the very last layers of the LLMs. Together with the weaker script-following performance observed in smaller models, these results form a converging body of evidence linking script commitment to model depth, with broader implications for the design of sufficiently deep, inclusive multilingual architectures.

1 Introduction

Large Language Models (LLMs) can generate text across scripts, from Latin and Cyrillic to Arabic and Devanagari (Team et al., 2022; Workshop et al., 2023; Hernández-Cano et al., 2026). This ability appears simple behaviorally, but internally, the model must first identify the script of the input, infer the script required for the output, and ultimately shift its probability mass toward tokens in the appropriate script. Transformer models process different aspects of linguistic knowledge at different depths (Tenney et al., 2019; Peters et al., 2018; Belinkov, 2022), as a consequence, these sub-tasks might not be executed simultaneously at a single point in the network, but instead distributed across layers.

This distributional assumption motivates our central research question: where, across the layers of an LLM, does script knowledge emerge, and does the answer differ depending on whether we consider the recognition of the input script, the planning of the output script, or the commitment of the output distribution to a target writing system? Prior work supports the plausibility of such layer-wise specialization beyond syntax and semantics. At the behavioral level, Liang et al. (2024) confirm that LLMs possess a degree of script planning capability, suggesting that output script selection is a deliberate, learnable operation rather than a mere surface artefact. At a coarser granularity, Pochinkov et al. (2024, 2025) show that commitment to paragraph-level content may be a general property of autoregressive generation, which we expect to extend to script-level commitment as well. Evidence further suggests that adjacent layers may not always build cooperatively on one another (Patrawala et al., 2025), that simpler tasks require fewer layers (Fan et al., 2025), and that high-resource language knowledge tends to be consolidated in deeper layers (Li et al., 2025).

Script processing, however, has not yet been studied from this perspective, even though this is a particularly interesting case: it intervenes at multiple stages of generation, from input recognition to output realization. Most closely related to our work is RomanLens (Saji et al., 2025), which pairs the logit lens with activation patching to study the mechanism of romanization and language transfer, and reports a related finding: intermediate layers often represent non-Roman-script targets in Romanized form before the model transitions to native script. Our work differs both methodologically, pairing the logit lens with linear probing rather than patching, and conceptually, distinguishing input script, instructed output script, and the script actually produced.

In this paper, separation of these three scripts is central and we investigate how script knowledge is distributed across the layers of large language models. We focus on the Qwen 2.5 Instruct family

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 24817–24825
October 24-29, 2026 ©2026 Association for Computational Linguistics

(0.5B–32B), which provides a controlled experimental setting.

To trace script processing through the model's layers, we employ two methods. In Section 3, we train probes (Alain and Bengio, 2017) on the hidden state of the final prompt token at every layer to identify where script information becomes linearly accessible, distinguishing input script, instructed output script, script produced by the LLM. Then, in Section 4, we apply a logit-lens analysis (nostalgebraist, 2020) to track when the output distribution commits its output distribution to the target script. Our experiments cover nine scripts[1]: Latin, Cyrillic, Arabic, Hebrew, Devanagari, Korean, Japanese, Chinese, and Armenian. Finally, showing that commitment to non-Latin scripts emerges only in the last layers of the LLM, in Section 5 we discuss this pattern as a potential, correlational explanation for the weaker performance of smaller models on non-Latin scripts.

2 Experimental Setup

2.1 Models

We work with the Qwen 2.5 Instruct family across six sizes: 0.5B, 1.5B, 3B, 7B, 14B, and 32B[2]. All checkpoints share the same pre-training corpus (Qwen et al., 2025), so behavioral differences across scales reflect architecture rather than knowledge. The models span 24 to 64 transformer layers, making this a good suite for studying the effect of depth. Note however that depth increases non-monotonically with parameter count: the 3B model (36 layers) is deeper than the 7B (28 layers). The full architecture table and quantization details are in Appendix A.

2.2 Data and Prompts

Prompts are drawn from nine source languages: English, Russian, Arabic, Chinese, Japanese, Korean, Hindi, Armenian, and Hebrew. Each language appears in two conditions: a canonical condition, where the model is instructed to produce output in the same script as the input, and a cross-script condition, where it is asked to use a different script. For all languages except English, the cross-script target is Latin. This yields 17 language-to-script mappings in total.

Within each mapping, prompts are drawn from three families: sentences, words, and questions. Each prompt is instantiated across four instruction variants that range from explicit (*"write the following using only Cyrillic characters"*) to concise (*"answer in Cyrillic"*). English and Russian are represented by 274 input items each (80 sentences, 114 words, 80 questions); the remaining seven languages by 70 items each (20/30/20). The full dataset comprises 8,304 prompts. Each model sees the same set. Detailed prompt templates are listed in Appendix B.

2.3 Script Classification

The script of a token is determined by the Unicode block name of its first character. We define ten script classes: one for each available input script and a "Neutral" class for tokens whose first character is any character that doesn't have a script. Tokens in the Neutral class are excluded from script-specific analysis. This classifier is applied at two points: to the prompt, to derive the input-script label, and to the first generated token[3] of each model response, to derive the used-output-script label.

2.4 Probing

For each (model, prompt) pair, we record the hidden state at the last prompt token at every layer (the embedding output from layer 0 and each of the $L-1$ transformer block outputs) extracted during the prefill step of the generation forward pass, without a second inference. We train one independent logistic regression probe per (target, layer) pair against three targets: *Input script (9 classes)*: the source script of the prompt, *Requested-output script (9 classes)*: the script the prompt instructs the model to use, and *Used-output script* (10 classes): the script the model actually produced, including a Neutral/Other bucket for non-classifiable outputs.

Probes are a logistic regression with ℓ_2 regularization trained on per-feature standardized activations. High accuracy from such a probe indicates that the target is geometrically accessible (hidden states of different classes are linearly separable in activation space, Hewitt and Liang, 2019) in the representation at that layer, not merely encoded in some nonlinear fashion. We use a 70/30 stratified train/test split, averaged across five independent random splits to reduce variance. Our primary met-

[1]Data is available on our GitHub repo https://github.com/IDSIA-NLP/LLMScriptCommitment

[2]The 14B and 32B checkpoints were run under 4-bit quantization

[3]We focus on the first generated token because it directly reflects the initial script-selection decision before autoregressive feedback effects

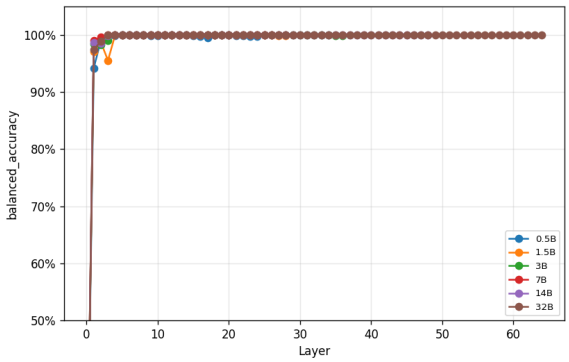

Figure 1: Balanced accuracy of the requested-output script probe across layers, for all six Qwen 2.5 model sizes.

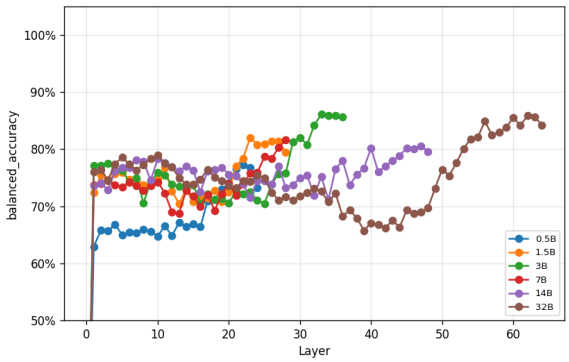

Figure 2: Balanced accuracy of the used-output script probe across layers.

ric is balanced accuracy (mean per-class recall), which is insensitive to the class imbalance introduced by the Neutral bucket in the used-output target; we also report AUC and raw accuracy as secondary metrics. Full hyperparameters are in Appendix C.

2.5 Logit Lens

The logit lens (nostalgebraist, 2020) projects each intermediate representation to obtain a distribution over the full vocabulary at every layer. We apply this to the same last-prompt-token hidden states used in section 2.4, requiring no additional forward pass. At each layer ℓ, we extract the top-1 token of the projected distribution and classify its script using the classifier of section 2.3. We define the script commitment layer as the earliest ℓ at which the top-1 token belongs to the target script[4]. We report two quantities: (i) the fraction of examples where the model's generated token belongs to the correct script, and (ii) the mean script commitment layer over successful examples, broken down by target script and model size.

3 Encoding of Script

At layer 0, all targets are at chance for every model ($\simeq 0.11$ for the 9-class targets and $\simeq 0.09$ for used-output), confirming that token embeddings carry no linearly accessible script information. This, in fact, is expected and serves as a sanity check, since all our prompts end with "Answer:" (see Appendix B) and, hence, have as the last prompt token always ":". However, at layer 1, input script already reaches $\simeq 0.98$ balanced accuracy across all model sizes.

[4]Examples without such layer are considered as failures

Figure 1 shows balanced accuracy for the requested-output target. All six models reach 1.00 by layer 3–5, regardless of size or depth: the script the model was instructed to use is fully encoded within the first $\sim$5–20% of the network. Notably, balanced accuracy decreases slightly in the final layers of each model suggesting that the explicit representation of the requested script is gradually displaced as the network shifts from encoding the instruction to computing the output distribution.

In Figure 2, the used-output target rises more slowly, peaks at 82–100% of model depth, and never saturates, with balanced accuracy ranging from 0.77 (0.5B) to 0.86 (3B) at peak. The maximum is set in part by each model's actual script-following rate (0.64–0.83 strict, across sizes). Together, the two figures establish the core dissociation: the model encodes what it should produce within the first few layers, but what it will actually produce only becomes readable in the final layers. Section 4 examines what happens in between.

4 Late Commitment to Output Script

Figure 3 shows the mean script commitment layer per target script and model size, computed over successful generations only. It shows an important contrast between Latin and all other scripts: the top-1 predicted token at the generation position commits to Latin within the first 1–2 layers across all six models regardless of the source language of the prompt. Non-Latin script generation requires a late override: for Cyrillic, Arabic, Devanagari, Korean, and Hebrew, the commitment layer consistently falls above 85% of model depth, often in the final three to five layers.

The commitment layer grows with model depth in absolute terms: for Cyrillic, it rises from $\sim$22

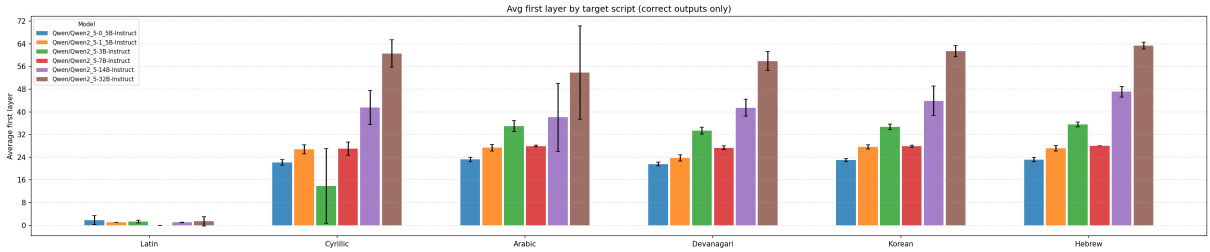

Figure 3: Average first layer of script commitment by target script and model size. Error bars show standard deviation across prompts. Hollow profiles show the maximum each model can reach (number of layers).

layers (0.5B, 24 total) to ∼60 layers (32B, 64 total), while remaining at roughly the same relative position across most models and scripts. One notable exception is Cyrillic for the 3B model, which commits at layer ∼14 out of 36. Armenian, Chinese, and Japanese yield too low success rates across all model sizes and are absent from Figure 3. Detailed results are in Appendix F, and show that when their output is the correct one, they exhibit the same late-commitment pattern as the other non-Latin scripts.

Layer	Prediction
h_out	' K'
h_{h34}_out	' K'
h_{h33}_out	'-capital'
h_{h32}_out	'-capital'
h_{h31}_out	'-capital'

Table 1: Logit-lens top-1 prediction at the generated token position across the last 5 layers of QWEN2.5-3B-INSTRUCT when asked to write "Capital" using the Cyrillic script. The ' K' character is a Cyrillic character.

Table 1 illustrates the underlying mechanism. The heatmap shows the top-1 logit-lens token at each layer and each prompt position for a representative English-to-Cyrillic generation (target: "Капитал"). Through most of the network, the top-1 token at the generation position is the semantically correct Latin token (*"capital"*). Only in the final five to eight layers does the prediction switch to the Cyrillic equivalent. The Cyrillic token is not arbitrary: it is the transliteration of the Latin prediction held in the preceding layers. This two-stage process accounts for both the late commitment observed in Figure 3 and suggests that the failure of models to complete it might be due to the insufficient model depth. Additionally, this finding corroborates the one of RomanLens (Saji et al., 2025), that independently reports a similar late-layer script transition on different models.

5 Models' Depth Bottleneck Hypothesis and Smaller Models Failures

Our results show that script encoding and identification are distributed across the model's layers. Input script identity is encoded at the first transformer block (layer 1, ∼98% balanced accuracy). The instructed output script is also captured and encoded within the first five layers (∼5–20% of depth). The vocabulary distribution, however, does not commit to non-Latin scripts until the final layers (≥85% of depth), and does so imperfectly in proportion to each model's depth. More specifically, Figure 4 presents the total probability mass assigned to Latin tokens, tokens in the requested target script, and tokens in any other script, at each layer. It shows that for most of each model's depth, Latin mass matches or exceeds target-script mass, with target-script mass surpassing Latin mass only toward the final layers. Qwen2.5-0.5B is the one exception, where Latin mass stays at or above target-script mass even at the very last layer, consistent with the observed model lack of depth.

Linear accessibility does not imply functional use The gap between layers of target script encoding and layers of realization in the distribution reflects a distinction between encoding information and using it. The model keeps the target script information accessible from layer 5 onward, yet its output distribution does not reflect this. We interpret the intervening computation as the network working through the semantic content of the answer in its default register (Latin), with script conversion deferred to the final layers. The slight decrease in requested-output probe accuracy observed at the end of the network (Section 3) is consistent with this view: as the final layers shift from maintaining the instruction to executing it, the explicit linear representation of the target script is partially displaced by the output distribution it is generating.

The non-monotonic relation between model size

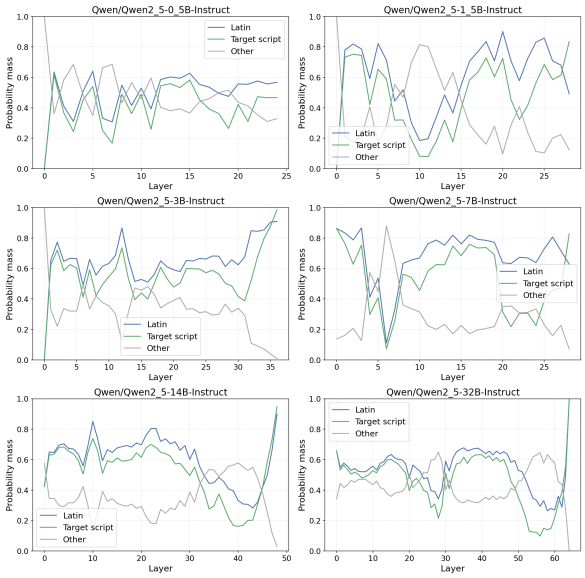

Figure 4: Probability mass assigned to tokens of Latin, target and any other script across layers.

in parameters and script-following performance shows that depth is the important parameter for this task. Non-Latin generation requires enough layers to execute the late-stage script conversion that our logit-lens analysis makes visible. A model with too few layers defaults to a neutral or Latin output, regardless of how well it has encoded the instruction.

We thus interpret the smaller models' struggle with non-Latin scripts as supportive of the hypothesis that limited model depth may constitute a bottleneck to completing the late-stage script conversion process, rather than reflecting a failure of knowledge or representation.

6 Conclusion

In this paper, we have shown that script processing in LLMs is not a monolithic operation. Rather, input identification, instruction encoding, and output script commitment emerge at different stages of the network. The output vocabulary distribution is modified only after the model has formed an internal representation of what it intends to produce, and the gap between representation and script realization is resolved only in the final layers.

The observed pattern offers a potential mechanistic explanation for the weaker script-following performance of smaller models: rather than failing to encode the target script instruction, it suggests that these models lack layers to complete the late conversion process.

As such, our findings have broader implications for the development of multilingual LLMs: improving support for underrepresented writing systems may require not only better representations, but also sufficiently deep architectures to translate those representations into reliable generation.

Future work should also investigate how LLMs acquire the ability to transliterate their top Latin prediction into the target script, and complement our script-based analysis with a breakdown by language family.

Limitations

We acknowledge the following limitations. First, all experiments are conducted on a single model family, Qwen 2.5 Instruct. It remains an open question whether the observed patterns generalize to other architectures. Second, the two largest models (14B and 32B) were run under 4-bit quantization due to hardware constraints. Third, Armenian, Chinese, and Japanese exhibited too low script-following success rates across all model sizes to be included in the main logit-lens analysis.

Finally, our probing and logit-lens analyzes both rely on a single token position: the last token of the prompt, which serves as the prediction site for the first generated token.

Acknowledgments

The work described in this paper has been partially funded by the "*Language Preference and Hallucination Investigation in Multilingual RAG (LPHI-mRAG)*" project, funded by armasuisse S&T, Switzerland.

References

Guillaume Alain and Yoshua Bengio. 2017. Understanding intermediate layers using linear classifier probes.

Yonatan Belinkov. 2022. Probing classifiers: Promises, shortcomings, and advances. *Computational Linguistics*, 48(1):207–219.

Siqi Fan, Xin Jiang, Xiang Li, Xuying Meng, Peng Han, Shuo Shang, Aixin Sun, and Yequan Wang. 2025. Not all layers of llms are necessary during inference. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '25.

Alejandro Hernández-Cano, Alexander Hägele, Allen Hao Huang, Angelika Romanou, Antoni-Joan Solergibert, Barna Pásztor, Bettina Messmer, Dhia Garbaya, Eduard Frank Ďurech, Ido Hakimi,

Juan Garcia Giraldo, Mete Ismayilzada, Negar Foroutan, Skander Moalla, Tiancheng Chen, Vinko Sabolčec, Yixuan Xu, Michael Aerni, Badr AlKhamissi, and 83 others. 2026. Apertus: Democratizing open and compliant LLMs for global language environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 46877–46955, San Diego, California, United States. Association for Computational Linguistics.

John Hewitt and Percy Liang. 2019. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 2733–2743, Hong Kong, China. Association for Computational Linguistics.

Daoyang Li, Haiyan Zhao, Qingcheng Zeng, and Mengnan Du. 2025. Exploring multilingual probing in large language models: A cross-language analysis. In *Proceedings of the 1st Joint Workshop on Large Language Models and Structure Modeling (XLLM 2025)*, pages 61–70, Vienna, Austria. Association for Computational Linguistics.

Sirui Liang, Baoli Zhang, Jun Zhao, and Kang Liu. 2024. ABSEval: An agent-based framework for script evaluation. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 12418–12434, Miami, Florida, USA. Association for Computational Linguistics.

nostalgebraist. 2020. interpreting GPT: the logit lens — LessWrong — lesswrong.com. https://www.lesswrong.com/posts/AcKRB8wDpdaN6v6ru/interpreting-gpt-the-logit-lens. [Accessed 24-05-2026].

Arjun Patrawala, Jiahai Feng, Erik Jones, and Jacob Steinhardt. 2025. Llm layers immediately correct each other. In *Advances in Neural Information Processing Systems*, volume 38, Main Conference, pages 139071–139113. Curran Associates, Inc.

Matthew E. Peters, Mark Neumann, Luke Zettlemoyer, and Wen-tau Yih. 2018. Dissecting contextual word embeddings: Architecture and representation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1499–1509, Brussels, Belgium. Association for Computational Linguistics.

Nicholas Pochinkov, Angelo Benoit, Lovkush Agarwal, Zainab Ali Majid, and Lucile Ter-Minassian. 2024. Extracting paragraphs from llm token activations. *Preprint*, arXiv:2409.06328.

Nicky Pochinkov, Yulia Volkova, Anna Vasileva, and Sai V R Chereddy. 2025. Parascopes: What do language models activations encode about future text? *Preprint*, arXiv:2511.00180.

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, and 25 others. 2025. Qwen2.5 technical report. *Preprint*, arXiv:2412.15115.

Alan Saji, Jaavid Aktar Husain, Thanmay Jayakumar, Raj Dabre, Anoop Kunchukuttan, and Ratish Puduppully. 2025. RomanLens: The role of latent Romanization in multilinguality in LLMs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 26410–26429, Vienna, Austria. Association for Computational Linguistics.

J. Ben Tamo, Daniel Carlander-Reuterfelt, Jonathan Rubin, Oleg Poliannikov, Dezhi Hong, and Mingxian Wang. 2026. Linguamap: Which layers of LLMs speak your language and how to tune them? In *The Fourteenth International Conference on Learning Representations*.

NLLB Team, Marta R. Costa-jussà, James Cross, Onur Çelebi, Maha Elbayad, Kenneth Heafield, Kevin Heffernan, Elahe Kalbassi, Janice Lam, Daniel Licht, Jean Maillard, Anna Sun, Skyler Wang, Guillaume Wenzek, Al Youngblood, Bapi Akula, Loic Barrault, Gabriel Mejia Gonzalez, Prangthip Hansanti, and 20 others. 2022. No language left behind: Scaling human-centered machine translation. *Preprint*, arXiv:2207.04672.

Ian Tenney, Dipanjan Das, and Ellie Pavlick. 2019. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601, Florence, Italy. Association for Computational Linguistics.

BigScience Workshop, :, Teven Le Scao, Angela Fan, Christopher Akiki, Ellie Pavlick, Suzana Ilić, Daniel Hesslow, Roman Castagné, Alexandra Sasha Luccioni, François Yvon, Matthias Gallé, Jonathan Tow, Alexander M. Rush, Stella Biderman, Albert Webson, Pawan Sasanka Ammanamanchi, Thomas Wang, Benoît Sagot, and 375 others. 2023. Bloom: A 176b-parameter open-access multilingual language model. *Preprint*, arXiv:2211.05100.

A Models

See details in Table 2. Large models (Qwen2.5-14B and Qwen2.5-32B) were loaded with 4-bit) quantization via BitsAndBytes, using double quantization and bfloat16 compute.

Model	Params	Layers	Hidden dim	Attn / KV heads
Qwen2.5-0.5B-Instruct	∼0.5B	24	896	14 / 2
Qwen2.5-1.5B-Instruct	∼1.5B	28	1536	12 / 2
Qwen2.5-3B-Instruct	∼3B	36	2048	16 / 2
Qwen2.5-7B-Instruct	∼7B	28	3584	28 / 4
Qwen2.5-14B-Instruct	∼14B	48	5120	40 / 8
Qwen2.5-32B-Instruct	∼32B	64	5120	40 / 8

Table 2: Qwen2.5 models used.

B Prompts

All prompts end by asking the answer with the word "`Answer:`" so that the model's first generated token is the start of the answer. Placeholders are shown in angle brackets: ⟨*target_script*⟩ is the name of the requested script (e.g. *Latin*, *Cyrillic*), and ⟨*input*⟩ is the source sentence, word, or question.

Sentences

Variant	Template
`write_using`	Write the following sentence using the ⟨*target_script*⟩ script: "⟨*sentence*⟩"
`rewrite`	Rewrite this sentence in ⟨*target_script*⟩: "⟨*sentence*⟩"
`write_chars`	Write the sentence "⟨*sentence*⟩" using ⟨*target_script*⟩ characters.
`how_write`	How would you write the following sentence in ⟨*target_script*⟩? "⟨*sentence*⟩"

Words

Variant	Template
`write_using`	Write the word "⟨*word*⟩" using the ⟨*target_script*⟩ script.
`write_in`	Write the following word in ⟨*target_script*⟩: ⟨*word*⟩
`how_write`	How do you write "⟨*word*⟩" in the ⟨*target_script*⟩ script?
`spell`	Spell the word "⟨*word*⟩" using ⟨*target_script*⟩ characters.

Questions

Variant	Template
`answer_using`	Answer the following question using the ⟨*target_script*⟩ script: ⟨*question*⟩
`inline`	⟨*question*⟩ (Answer using the ⟨*target_script*⟩ script.)
`terse`	Answer in ⟨*target_script*⟩: ⟨*question*⟩
`only_chars`	Respond to the following question using only ⟨*target_script*⟩ characters: ⟨*question*⟩

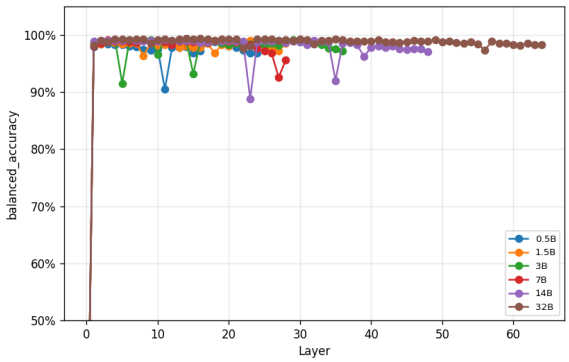

Figure 5: Balanced accuracy of the input-language probe across layers, for all six Qwen models.

C Probes

ℓ_2 regularisation ($C = 1$), and L-BFGS optimisation, on a single 70/30 train/test split of 1,000 prompts after per-feature standardisation.

D Script-Following Behavior

We check each model's behavioral compliance with the requested script. The strict follow rate is the fraction of prompts for which the used-output script matches the requested script; the lax follow rate additionally counts "Other" and "Neutral" outputs as successes (Table 3).

Model	Strict	Lax
Qwen2.5-0.5B-Instruct	0.641	0.797
Qwen2.5-1.5B-Instruct	0.719	0.846
Qwen2.5-3B-Instruct	0.831	0.920
Qwen2.5-7B-Instruct	0.799	0.898
Qwen2.5-14B-Instruct	0.766	0.838
Qwen2.5-32B-Instruct	0.754	0.833

Table 3: Script-following rates.

E Input language

Input language is the easiest target by a wide margin: it is fully determined by the surface tokens of the prompt, requiring no cross-token integration. At layer 0 (embedding output) all probes sit at chance (balanced accuracy 0.47–0.54); a single transformer block suffices to drive every metric to ≈1.0 in every model.

F Detailed Results

Figure 6 illustrates per-script breakdown across all six Qwen models.

To investigate the distribution of probability mass beyond the top-1 token at each layer, we cal-

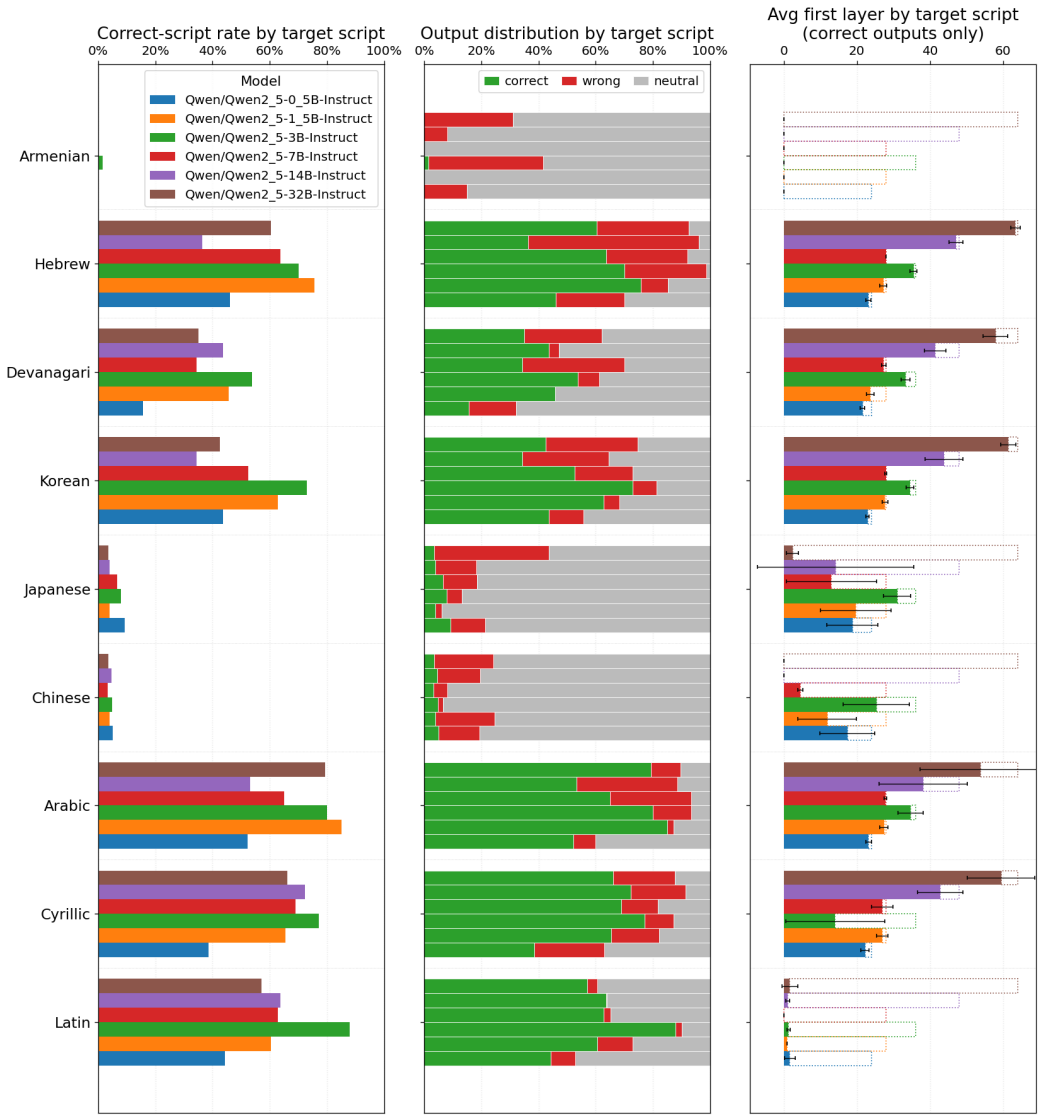

Figure 6: Per-script breakdown across all six Qwen models. Left: correct-script generation rate by target script. Center: output distribution by target script, broken down into correct, wrong, and neutral outputs. Right: average first commitment layer by target script for correct outputs only.

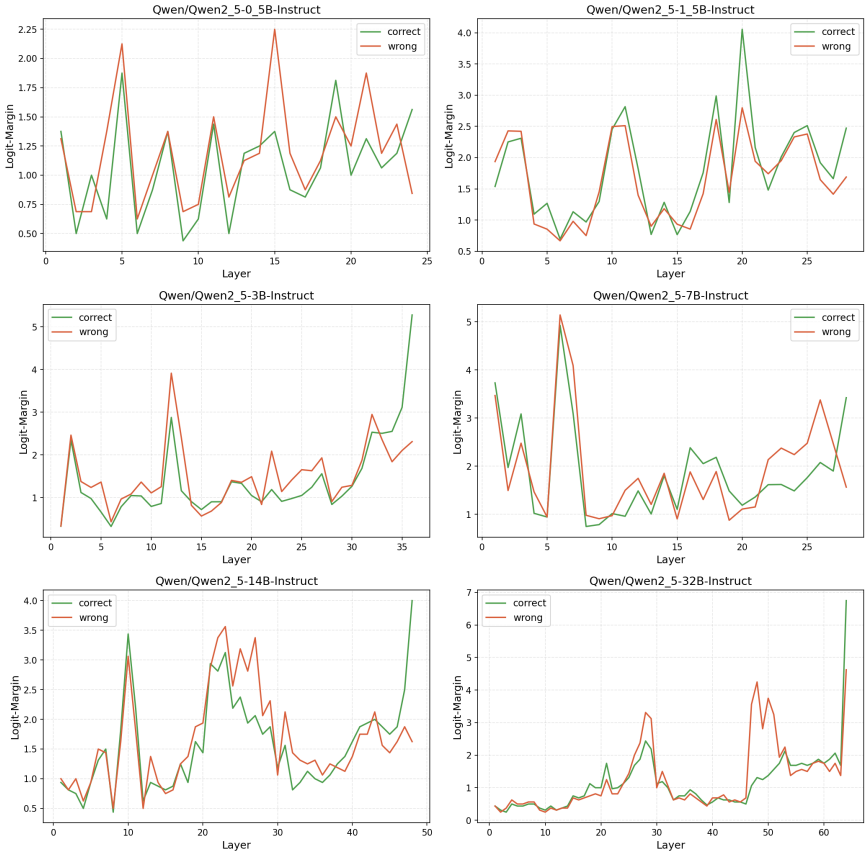

Figure 7: Logit margin between the top-1 token and the highest-ranked token from different script, across layers.

culate the logit margin between the top-1 token and the highest-ranked token belonging to a different script (Figure 7). Across all six models, the logit margin stays modest (from 0.5 to 5), before increasing in the final layers, with the two models 14B and 32B showing the largest increase in margin at the final layer. This confirms top-1 at the commitment layer is not a weak tie-break. At the last layer, prompts whose output matches the requested script show a larger margin than those that do not.

G Wrong Scripts Generation

For each model, we add in Figure 8 a breakdown of the prompts whose used-output script does not match the requested script, by the script that was actually produced.

H Computational Details

All experiments were run on an NVIDIA L40S GPU (46 GB VRAM). All models were run with greedy decoding (`do_sample=False`) and a maximum budget of 256 new tokens. Generation was stopped at the first token whose decoded text contained a classifiable Unicode character.

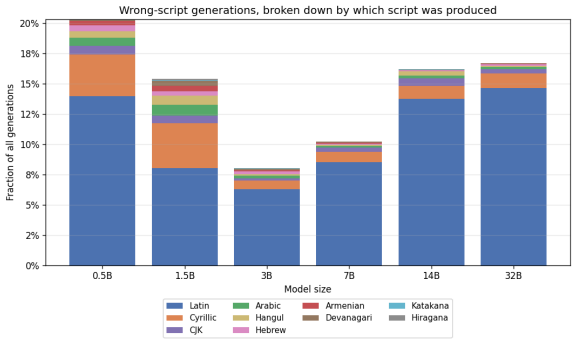

Figure 8: Breakdown of wrong-script generations by the script actually produced, per model.

Probes were trained on GPU using a batched logistic regression implemented in PyTorch, fitting all layers of a given target simultaneously in a single Adam optimization loop (learning rate 0.1, up to 1000 epochs).