OpenCódice at PoliticHeadlinES-IberLEF 2026: An Agentic Approach to Multimodal Political-Headline Ranking

Francisco-Javier Rodrigo-Ginés[1,*], Jorge Chamorro-Padial[2]

[1]Universidad Nacional de Educación a Distancia (UNED), 28040 Madrid, Spain
[2]OpenCódice Research, Spain

Abstract

We describe the OpenCódice submission to PoliticHeadlinES 2026 at IberLEF, in which the experimentation pipeline was driven by an agentic research loop. A coding agent built on Claude Opus 4.7 received the task description, the raw data, GPU access and API credentials, and iterated through a sequence of ranking strategies under researcher supervision. We position the work in the autoresearch space recently popularised by Karpathy [1], but argue, with empirical evidence from this campaign, that a researcher-in-the-loop variant is preferable for shared-task participation: the agent proposes, executes and analyses; the researcher gates submissions and direction changes. The agent implemented nine substantially different configurations, ranging from zero-shot GPT-5.4 listwise prompting and a pointwise XLM-RoBERTa-large ranker to gradient-boosted models on hand-crafted features, multilingual sentence-embedding similarity, vision-language post-hoc reranking with Qwen2-VL, and Borda-count ensembles. The five strongest of these produced identical top-1 picks for every one of the 4 912 test articles, despite differing substantially on the rest of the ranking, and every one of them scored within ±0.001 of 0.884 on the official position-aware nDCG@K metric. We treat this empirical convergence as the most useful finding for future participants. Our submission ranked seventh of nineteen participating teams. The position we defend, with operational evidence from the campaign, is that agentic AI dramatically accelerates empirical NLP research, but that hypothesis generation and the acceptance criterion belong, irreducibly, to the human researcher.

Keywords

Multimodal headline ranking, Agentic research, Autoresearch, Human-in-the-loop, Spanish political news

1. Introduction

Can a coding agent, given a shared-task description, the data, a GPU account and an API key, run an end-to-end empirical NLP campaign? The question is no longer speculative. Foundation-model assistants have become competent enough at code synthesis [2], at resolving real software-engineering tasks at the level of GitHub issues [3], and at chaining tool use, planning and error recovery [4, 5], that the configuration sketched by Karpathy as *autoresearch* [1], an agent that proposes a direction, writes the code, executes it, reads the results and updates its plan, is no longer aspirational but a working configuration that one can deploy on a laptop. Recent demonstrations cover machine-learning experimentation [6], fully automated scientific paper drafting [7] and autonomous chemistry research [8]. What the field still lacks is calibration: end-to-end evidence about what such an agent actually produces in front of an external leaderboard, where the metric is inarguable, the deadline is hard, and the score returned by the organisers cannot be spun. Shared-task participation is, in our view, the cleanest available calibration ground for that question. The scope is bounded, the iteration loop is short, the data are fixed, and the leaderboard is impossible to argue with.

This paper is one such calibration data point. The OpenCódice team entered PoliticHeadlinES 2026 [9], part of the IberLEF 2026 forum [10], and operated its experimentation pipeline through a Claude Opus 4.7 coding agent under researcher supervision. The task, defined on the Spanish PoliSUM-2025 dataset [11], asks a system to rank ten Spanish political headlines for a news article body and its accompanying image: exactly one candidate is the originally published headline and the rest are

IberLEF 2026, September 2026, León, Spain
*Corresponding author.
frodrigo@invi.uned.es (F. Rodrigo-Ginés); jorge@opencodice.org (J. Chamorro-Padial)

paraphrase distractors or off-topic headlines from the same corpus. Two subtasks are defined, one text-only (Task 1) and one multimodal (Task 2), and both are scored with a position-aware variant of normalised discounted cumulative gain. Nineteen teams participated.

The agent implemented every training and inference script in the campaign, launched them on remote GPU instances, parsed their logs, constructed submission files, and revised its working hypotheses in light of the leaderboard returns. The researcher's contribution was concentrated at the two ends of the campaign. At the start, the researcher specified the methodological framework: the decision to adopt a researcher-in-the-loop protocol rather than fully autonomous autoresearch, the staged plan that moved from exploratory data analysis to baseline construction, then to single-model fine-tuning and finally to multi-component fusion, the resource budget allocated to each strategy class, and the acceptance criteria under which a configuration would be promoted to a submission candidate. During the campaign, the researcher exercised a small and disciplined set of veto points: every CodaBench upload required explicit approval, and any non-trivial change of strategic direction required prior endorsement. Between those two boundary conditions, the operational execution of the experimental loop was carried out by the agent. We refer to this configuration as *researcher-in-the-loop* agentic research and document it in operational detail in the remainder of the paper.

The central thesis of this work, which the rest of the paper defends with operational evidence from the campaign, is twofold. First, we believe that agentic AI accelerates empirical NLP research by an order of magnitude or more on the time scale of the next few IberLEF editions. Most of the wall-clock cost of shared-task participation is no longer time spent reasoning about the problem but time spent on its mechanical execution: writing the training loop, parsing the log, fixing the parser bug, constructing the next prediction file. A capable coding agent compresses execution work that previously occupied a researcher for a full working day into a small fraction of that interval, and does so in parallel with the researcher's own deliberation on the next experiment. Second, and equally important, we believe that this acceleration is bounded by two functions that do not transfer to the agent, and that we expect will not transfer on the time scale of this campaign nor the next. Those two functions are *hypothesis generation*, in the sense of deciding which research direction is worth pursuing, and the *acceptance criterion*, in the sense of deciding what counts as a successful result on the way to the final claim. The first has been the subject of recent empirical study at the idea-generation level [12], with mixed evidence about LLM novelty and a clear gap on feasibility; the second is, in our view, an even harder transfer problem because it requires the kind of context that is rarely written down. Both remain, in our view, irreducibly the researcher's responsibility. The agent proposes implementations and reports outcomes; the researcher decides what to investigate and what counts as evidence. The intervention is rare in any given iteration; the cost of skipping it is, in our experience, several wasted days per dead-end.

The campaign produced two findings that motivated writing this paper rather than merely uploading the submission.

The first finding is methodological. The researcher veto, although exercised rarely, caught failure modes that the autonomous loop would not have caught on its own. Two of these failure modes recur in agentic research practice and we describe them in Section 4.6: a *sunk-cost* attachment to an early ranker that the agent kept refining well past the point of diminishing returns, and a *metric-literalism* mistake, a near-cousin of the reward-hacking failure mode formally characterised by Skalse and colleagues [13], in which the agent interpreted "ranking" as a full listwise ordering problem rather than as the top-heavy near-top-1 indicator that the official scoring program actually computes. Each was resolved in a single sentence, after the researcher asked the agent to defend its current direction.

The second finding is empirical and is the substantive technical result of the paper. The agent explored nine substantially different ranking strategies during the campaign: zero-shot listwise prompting with GPT-5.4 and GPT-5.5, in the style recently popularised for retrieval reranking by Sun et al. [14] and Pradeep et al. [15], a pointwise XLM-RoBERTa-large ranker, a gradient-boosted model on hand-crafted features, sentence-embedding similarity baselines, a vision-aware listwise prompt, a post-hoc reranker driven by Qwen2-VL image captions, a Borda-count ensemble, and a score-level fusion of the three strongest components. The five strongest of these strategies produced *identical top-1 picks* on every one of the 4 912 test articles, despite differing substantially on positions 2 to 10. Every one of the five

landed within ±0.001 of 0.884 on nDCG@10 on both subtasks. Our submission scored 0.884 and ranked seventh of nineteen participating teams.

We read this convergence as direct evidence that PoliticHeadlinES is not, in practice, a listwise ranking problem at all. The metric collapses on this dataset to a top-1 indicator with a small contribution from the rest of the ordering, and the natural reformulation of the task is therefore top-1 classification with near-paraphrase distractors. Future participants who close the gap to the leading systems will, we expect, do so by discriminating better between a gold headline and its closest paraphrase decoy, not by improving the rest of the ranking.

Roadmap. Section 2 positions the work in the agentic-research literature and the literature on top-heavy ranking metrics. Section 3 describes the task formally and presents the exploratory analysis that drives the system design. Section 4 is the central methodological contribution: the loop, the autonomy spectrum that frames it, the two safety gates, and the failure modes we observed. Section 5 describes the submitted system. Section 6 reports every strategy the agent explored and the convergence pattern that motivated this paper. Section 7 draws the transferable lessons for future agentic submissions, and Section 8 concludes.

2. Related Work

The work belongs to the intersection of three lines of research: agentic research with language models, human-in-the-loop machine learning, and position-aware ranking metrics for information retrieval. We sketch each in turn, with emphasis on the autonomy spectrum that frames our methodological choice.

The line of work most directly relevant to our framing is the recent push to operate language models as autonomous research agents. Voyager [16] demonstrated that an LLM agent can drive an open-ended exploration of a complex environment without an externally specified curriculum. The AgentBench benchmarks [17] formalised LLM-as-agent evaluation across a wide variety of tasks, including database and operating-system interaction. The broader survey of LLM-based autonomous agents by Wang and colleagues [18] catalogues the design choices that have emerged in the field: profiling, memory, planning and action modules, together with single-agent and multi-agent architectures for general problem solving; the parallel survey by Durante and colleagues [19] traces the multimodal extension of the same picture, which becomes relevant for the vision-aware component of our system. Anthropic's recent *computer-use* interface [20] extends agentic capability to the desktop, with the human in an auditing role rather than as the operator of every individual command. Karpathy's *autoresearch* [1] sketches a research loop in which the agent autonomously proposes a research direction, writes the code, runs experiments and reports results, treating the human researcher as an inspector of conclusions rather than as a per-step operator. Empirical instances of that loop have started to appear at scale. The *AI Scientist* of Lu and colleagues [7] drives an end-to-end paper-writing pipeline that ideates, codes, runs and drafts, in domains as varied as 2D diffusion and grokking, at a cost of about a dozen US dollars per manuscript. *MLAgentBench* [6] benchmarks LLM agents on thirteen machine-learning experimentation tasks under ReAct-style harnesses and reports headline success rates that range from full success on legacy datasets to zero on post-cutoff Kaggle problems, a profile that mirrors the failure modes we observed during this campaign. Outside NLP and ML, Boiko and colleagues [8] run an autonomous chemistry agent end to end on a robotic laboratory, including reaction planning, synthesis, and reporting, with no human in the per-experiment loop. Our methodology accepts the same premise about per-experiment autonomy, but pushes back against full automation of the loop on the basis of evidence that we present in Section 4.6: agents inherit failure modes from their underlying language model, and a researcher operating with strictly different priors is the cheapest available defence against them. The point about hypothesis generation in particular is informed by Si et al. [12], who report in a large human study that LLM-generated NLP research ideas are judged more novel than expert ideas but weaker on feasibility, which is consistent with our experience.

Within that agentic stack, the lineage of techniques used to interleave reasoning with acting deserves a

more careful treatment because we relied on each of them, formally or informally, during the campaign. ReAct [4] established the basic pattern of alternating natural-language reasoning steps with tool invocations, and remains the dominant shape of every agentic loop in production today, including the Claude Code harness used here. Reflexion [5] adds a verbal self-feedback loop that lets an agent learn from its mistakes within an episode without weight updates; we used the same pattern informally whenever the agent re-read its own transcript before proposing a follow-up experiment. Tree of Thoughts [21] generalises chain-of-thought reasoning [22] to a search procedure over alternative reasoning paths, and matters here because the agent's strategy proposals during this campaign were closer to a search over a tree of candidate research directions than to a linear chain. Toolformer [23] predates these patterns and demonstrated that language models can learn to invoke tools through self-supervision, anticipating the explicit tool-use formats now standard in commercial APIs. None of these techniques was applied to PoliticHeadlinES as a research contribution; we list them because they are the conceptual stack on which the agent's behaviour is most naturally described, and identifying which pattern the agent was using in a given moment is part of the reproducibility story for an agentic submission.

Researcher-in-the-loop machine learning has a much longer history that does not need rehearsing here. The relevant point for this paper is that the classical motivation for putting a human in the loop, that of cheaply correcting model outputs at the boundaries of the training distribution, is not the motivation that applies to our setting. The agents we work with are not corrected on individual predictions; they are corrected on entire research strategies. The intervention is, therefore, much rarer and much more impactful than the per-instance human review studied in classical active-learning settings. We see this as a small but genuine adaptation of the human-in-the-loop pattern to agentic research.

A complementary thread of work that mattered for the campaign is the emerging ecosystem of *Model Context Protocol* (MCP) servers [24]. MCP is a standardised tool-use interface that exposes external resources (databases, APIs, file systems, paper repositories) as structured tools that an LLM agent can invoke during its reasoning loop. Three MCP servers were active during the experimentation phase of this campaign and shaped which models and techniques the agent actually considered for the task. The ACL Anthology MCP server [25] exposed the ACL bibliography as a queryable resource, used by the agent to retrieve state-of-the-art Spanish text encoders, learning-to-rank methods, and multilingual encoder comparisons; it informed our choice of XLM-RoBERTa-large over the base variant and the decision to fall back to LightGBM features as a robustness check. The OpenReview MCP server [26] extended the same capability to NeurIPS, ICLR and adjacent venues, where much of the agentic ML and multimodal alignment literature is published; the agent queried it when deciding between Qwen2-VL and LLaVA for the post-hoc multimodal experiment. The Semantic Scholar MCP server [27] provided citation-graph traversal, used by the agent to expand from a seed paper (such as the original XLM-RoBERTa release) to recent follow-ups that informed training-recipe defaults. The use of MCP servers as a literature-grounding layer is a small but important methodological detail: it kept the related-work survey in this paper anchored to actual indexed publications rather than to the agent's training-cutoff memory, which has proven prone to hallucinated citations in our own earlier work and is, in our view, the single largest reproducibility risk in agentic paper writing today.

The Spanish-language ranker also sits in the active line on Spanish-specific NLP. BETO [28] and BERTIN [29] are the two most widely used monolingual Spanish encoders; BERTIN is competitive with multilingual encoders on classification tasks while requiring substantially fewer GPU hours to pre-train, and is the natural default for monolingual Spanish text classification. For PoliticHeadlinES we ultimately preferred XLM-RoBERTa-large [30], an evolution of the multilingual encoder family characterised on cross-lingual benchmarks such as XNLI [31] and routinely complemented by sentence-level encoders trained with the SBERT [32] recipe for retrieval-style tasks. The choice of a multilingual encoder for a monolingual Spanish task is deliberate: the candidate headlines and the article bodies in PoliticHeadlinES are sometimes near-translations of agency wire copy, and the multilingual representation tolerates that variation better than the monolingual encoders we tried in early ablations.

For the multimodal subtask we drew on the line of vision-language alignment work that begins

with CLIP [33] and its dual-encoder formulation, where image and caption embeddings are pulled together by a contrastive objective. BLIP [34] extended that formulation with a generative captioning head, which in turn anticipated the multimodal large language models such as Qwen2-VL [35] that we use as the captioning backbone in the post-hoc reranking experiment (Section 6). The trajectory from contrastive dual encoders to autoregressive vision-language models is the relevant context for understanding why a single Qwen2-VL caption pass is a viable substitute, on this task, for an explicit text-image similarity model.

The third line of work that frames our paper is the literature on top-heavy ranking metrics. The classical normalised discounted cumulative gain [36] weights every position by a logarithmic discount, which is already top-heavy by design. Production search systems often steepen that discount further to reflect the empirical observation that real users almost never read past the first few results. The learning-to-rank literature beyond DCG includes the LambdaRank/LambdaMART family of pairwise and listwise gradient-boosted rankers [37], which we use in our second component (Section 5). For tasks with a single relevant document and a steep discount, the metric collapses to something close to a top-1 indicator with a small contribution from the rest. We observed an empirical signature of exactly this collapse in PoliticHeadlinES 2026 (Section 6). We did not attempt to read the metric off its analytic form; rather, we characterise it empirically through the convergence pattern of five very different rankers on the test set.

3. Task, Data, and Exploratory Analysis

This section describes the task in formal terms, presents the splits and basic statistics of the dataset, and then walks through an exploratory analysis whose findings inform the design of the submitted system. The exploratory analysis is also the empirical record against which we will, in Section 6, compare the predictions of every strategy the agent tried.

3.1. Task definition and splits

Each instance of PoliticHeadlinES 2026 consists of a Spanish political news article body b, an associated image g, and ten candidate headlines $t_1, \dots, t_{10}$. Exactly one of the candidates is the original published headline; the remaining nine are either lexical perturbations of the original (paraphrase distractors) or unrelated headlines drawn from the same corpus (off-topic distractors). A submission must produce a permutation of $\{t_1, \dots, t_{10}\}$ ordered from most to least faithful to the article. The official metric is a position-aware variant of nDCG, which we will refer to throughout as nDCG@10, with one relevant document (the original headline) and nine irrelevant candidates per instance.

The dataset is partitioned into three splits: a training set of 16 030 fully labelled instances, a small public development set of 50 instances, and the public test set of 4 912 instances on which submissions are scored. The training set is large enough to support fine-tuning a transformer text encoder and a gradient boosted ranker on hand-crafted features, both of which we use in our system. The development partition, by contrast, is small enough that any system tuned aggressively on it risks distributional artefacts; we revisit this concern in Section 6.

3.2. Lengths and composition

Article bodies span a wide length range, from one-paragraph news flashes to long editorial pieces. Candidate headlines, in contrast, occupy a narrow band that is typical of newspaper publication. Figure 1 shows both distributions on the test set. The asymmetry has practical consequences for the input encoders used by our system: the body must in many cases be truncated to fit within a reasonable transformer context window, while the candidate headlines almost always fit comfortably.

The asymmetry also informs the system design choice between encoder-side and decoder-side modelling. A pointwise transformer ranker that consumes the body and candidate as a single sequence must allocate most of its context to the body, which in turn means the candidate is encoded with the

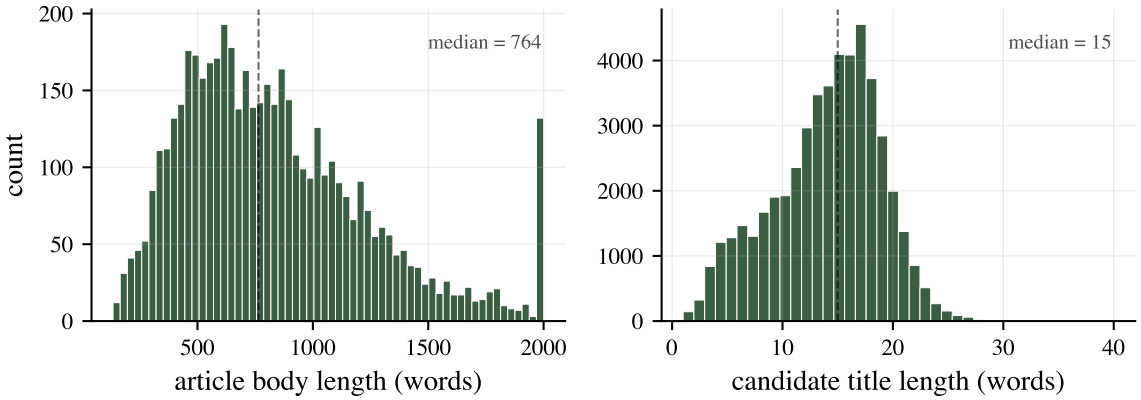

Figure 1: Length distributions on the test set. Left: article body length in words, clipped at 2 000 for readability; very few articles exceed this length. Right: candidate headline length in words across all 4 912 × 10 candidates. The median article is roughly an order of magnitude longer than the median candidate.

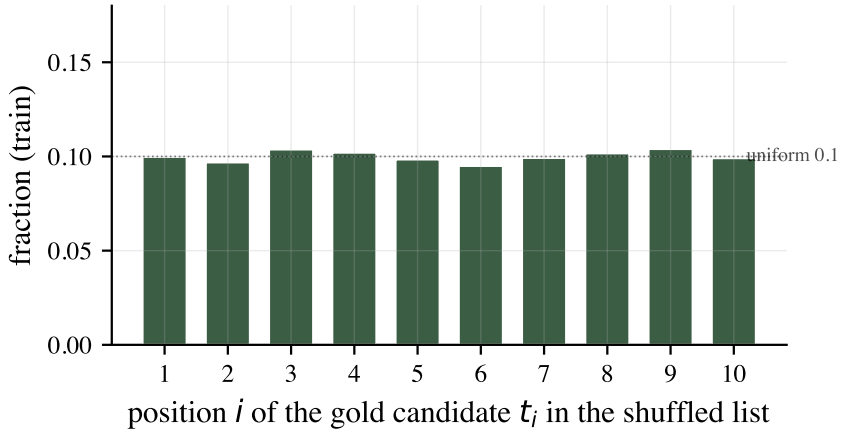

Figure 2: Position of the gold candidate within the shuffled list, on the training set. The dotted line marks the uniform expectation 1/10. The position carries no information by itself.

body almost entirely as contextual ballast. By contrast, a listwise prompted LLM that treats the body as background and the ten candidates as foreground sees the candidates side-by-side, which is closer to how a human reader compares them. We use both architectures in our system, in part because they are sensitive to this asymmetry in opposite ways and we expected the resulting predictions to be complementary; in Section 6 we will see that they are not, at least at the position the metric most cares about.

3.3. Position of the gold candidate

The candidate list of each article is shuffled independently. As a consequence, the position $i \in \{1, \dots, 10\}$ that holds the gold candidate is approximately uniform across the training set, with each i occurring as the gold position in roughly ten per cent of articles. Figure 2 shows the empirical distribution; it is uniform up to sample noise and contains no exploitable systematic bias. This is a deliberate design choice on the part of the organisers, and the consequence for system design is that any ranker that relies on candidate position as a feature, including any model that secretly memorises "the first option is usually wrong", will not generalise.

3.4. Paraphrase decoys are the discriminator

The most operationally important property of the dataset is the density of paraphrase distractors. Many of the nine non-gold candidates of a given article are off-topic and trivially distinguishable from the

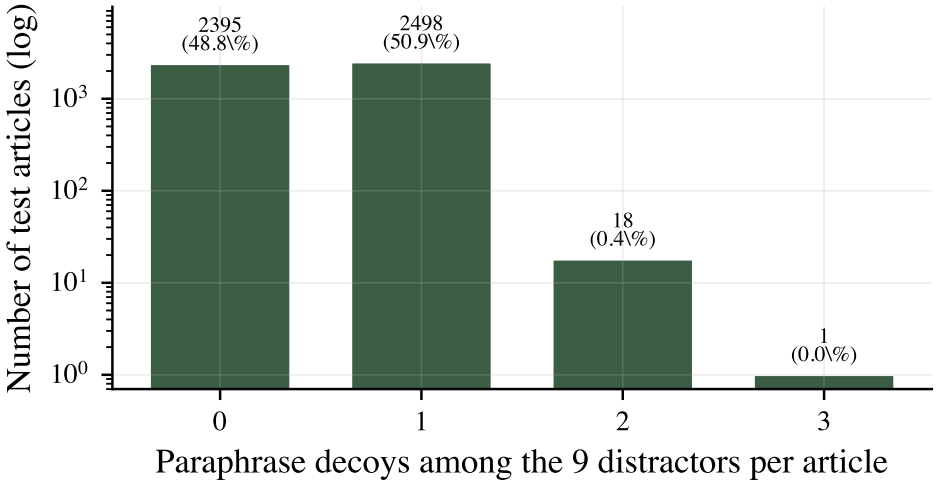

Figure 3: Paraphrase-decoy density per test article. The bars report the count of paraphrase distractors (token Jaccard ≥ 0.65 with the gold) among the nine non-gold candidates of each article; the y-axis is on a logarithmic scale so that the long tail (18 articles with two paraphrase decoys, one with three) is visible alongside the two dominant buckets. Roughly half of all test articles contain at least one such paraphrase. These are the discrimination cases on which a ranker actually has to do work.

gold; the discrimination problem only arises when one or more of the distractors share the bulk of the gold's content with a small lexical perturbation. We measured the phenomenon directly on the test set. For each article, we counted how many of the nine distractor candidates share at least 0.65 Jaccard overlap of word tokens with the gold candidate, a threshold that captures the case in which the distractor is syntactically close enough to the gold that lexical overlap is not, by itself, sufficient to identify the original. Figure 3 reports the result.

The dataset is, by this measure, bimodal. About half the articles have no paraphrase decoy in their candidate list, in which case any reasonable ranker will pick the gold candidate correctly on the basis of crude lexical overlap with the body. The other half contain at least one near-paraphrase distractor, and these are the articles on which the leaderboard is decided. Section 6 will revisit this observation, because the strategies our agent explored agree on every easy case and disagree only on the positions of the hard cases that the metric weights weakly.

4. Agentic Methodology

This section describes the researcher-in-the-loop methodology in operational detail. We open with a description of the loop and the roles within it, continue with a positioning of the loop on the spectrum of agentic autonomy, formalise the decision protocol with two safety gates, and close with a catalogue of the failure modes we observed during the campaign.

4.1. The researcher-and-agent loop

The submission was produced inside a tightly coupled loop with two actors. The agent is a Claude Opus 4.7 model [38] exposed to a working directory through a tool-use interface. It can read and edit files, execute shell commands, run Python and shell scripts, query OpenAI and OpenRouter HTTP endpoints, and download artefacts from CodaBench. The loop is seeded by the researcher: before the agent runs its first experiment, the researcher formulates the initial set of working hypotheses that frame the campaign, including the choice of candidate model families to consider, the expected sources of discriminative signal in the data, and the prior beliefs about which strategy classes are likely to be competitive. Each iteration of the loop is, in effect, an agent-driven probe of one of those seed hypotheses, refined by the evidence the previous iteration produced. The researcher reviews the agent's

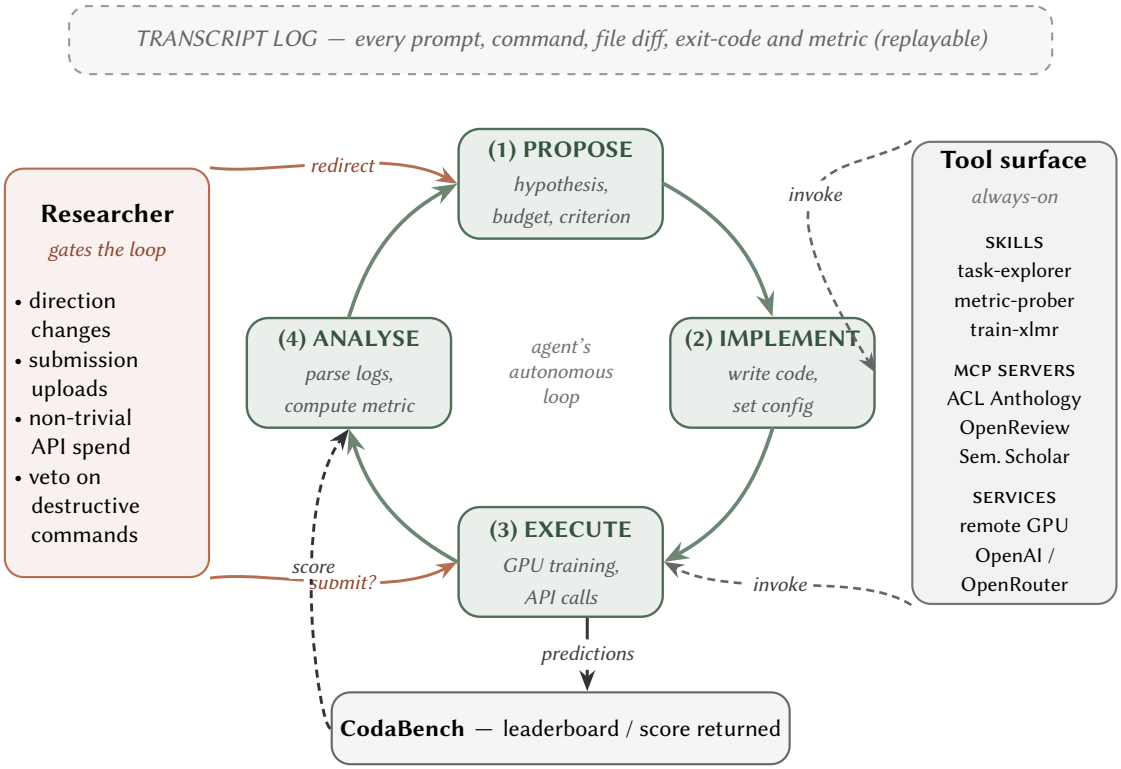

Figure 4: The researcher-and-agent research loop. The agent autonomously proposes, codes, runs and analyses experiments; the researcher gates submissions and direction changes. Every command issued by the agent is recorded in a transcript log that lives alongside the working repository.

natural-language proposals at every iteration, makes the final call on which experiment runs, decides which prediction file is uploaded to CodaBench, and intervenes when a strategy needs to be redirected or abandoned. Figure 4 renders the loop graphically.

A typical iteration of the loop proceeded as follows. The agent inspected the current state of the working directory, read the most recent experimental notes, and proposed a next experiment in natural language, including a hypothesis to test, the resources required, and the metric on which the result would be reported. The researcher accepted, modified, or rejected the proposal. In practice, the agent's first proposal was accepted unchanged in the majority of iterations; the researcher's modifications were typically about scope or risk rather than about scientific direction. Once accepted, the agent wrote the code, launched it locally or on a remote GPU, watched the logs, and produced a metric report at the end of the run. Every step of the iteration, including the natural-language reasoning, the tool calls and the shell output, was recorded in a transcript file shared with the working repository. The researcher then reviewed the report and, if the iteration produced a candidate submission, decided whether to upload it. The leaderboard score returned by CodaBench was pasted back into the agent's context, and the next iteration began.

The iterations had highly variable wall-clock duration. The shortest were a handful of minutes, when the agent reproduced an existing pipeline with a hyper-parameter change and the run fitted on a single GPU. The longest were several hours, when the agent shipped multimodal embeddings of the entire test set or fine-tuned a large encoder for several epochs. The researcher's work, by contrast, was concentrated at the start and end of each iteration, with the experimental work itself running unattended.

Table 1 makes the division of labour explicit across the five stages of a campaign iteration. The pattern is consistent: the agent owns implementation and execution, the researcher owns the two stages that decide what is worth doing and what counts as a result, hypothesis formulation and the interpretation of the analysis, and the externally visible action of submitting. Even at the stages each party leads, the other contributes: the researcher debugs and monitors the agent's runs, and the agent

Table 1
Division of responsibility between the agent and the researcher across the five stages of a campaign iteration. "Lead" marks the actor who drives the stage; "support" marks the actor who reviews, constrains or approves.

Stage	Agent	Researcher
Hypothesis formulation	support (suggests alternatives)	lead (proposes, prioritises)
Implementation	lead (writes code)	support (debugs)
Execution	lead (runs)	support (monitors)
Analysis	support (extracts basic statistics)	lead (interprets)
Submission	support (validates)	lead (approves, uploads)

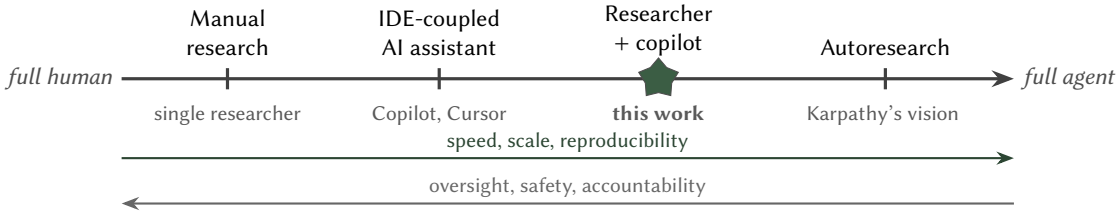

Figure 5: The spectrum of agentic autonomy in research. The horizontal axis ranges from fully manual research at the left to fully autonomous autoresearch at the right. Our position, marked with the star, gives the agent terminal autonomy and reserves submission and direction-change authority for the researcher. The two trade-off arrows below the axis sketch the intuition: as autonomy grows, speed and reproducibility grow with it, while oversight, safety and accountability decrease.

extracts the basic statistics on which the researcher's interpretation rests.

The support roles played by the agent are not symmetric with those played by the researcher. The agent's support is mechanical: it suggests alternative directions, extracts summary statistics, or validates a submission file against the local evaluator. The researcher's support is evaluative: deciding whether the agent's analysis warrants a submission, and whether a proposed direction is worth the compute. This asymmetry is the operational form of the central thesis of Section 1, that hypothesis generation and the acceptance criterion remain the researcher's responsibility.

4.2. Position on the autonomy spectrum

The researcher-and-agent loop is not the only way to organise LLM-driven research. Figure 5 sketches a spectrum of autonomy options, from fully manual research at the left to fully autonomous *autoresearch*, in Karpathy's sense, at the right. The intermediate positions are populated. IDE-coupled AI assistants such as Copilot and Cursor live near the left end: the human researcher writes most of the code and uses the AI to accelerate familiar steps. Computer-use agents [20] push further to the right, with the human acting as auditor of multi-step plans. Our position sits in between, closer to autoresearch than to IDE assistance: the agent runs experiments end-to-end without human supervision of individual steps, but the researcher retains authority over externally visible actions (submissions) and direction changes.

We chose this position deliberately. Pure autoresearch on a shared task has, by construction, no internal signal that warns the agent when it is optimising the wrong objective. If the agent misreads what the metric rewards (and we will report below that ours did), no internal feedback will steer it back. A researcher operating with strictly different priors and reading the leaderboard scores at the natural-language level is the cheapest defence against that kind of error. The same argument applies, to a lesser degree, against fully autonomous strategy choices: agents inherit failure modes from their underlying language model, and those failure modes include sunk-cost attachment to a strategy once compute has been invested. A researcher in the loop is, on those occasions, much cheaper than a wasted period of compute.

The pilot–copilot framing is a useful frame for the position we adopt. In conventional copilot setups the human writes most of the code and the model accelerates familiar steps; in autoresearch the roles

are reversed and the human becomes a remote inspector of conclusions. Our setup keeps the human as the *pilot* for strategic and externally visible decisions while letting the agent function as the *flying officer* who runs the aircraft moment to moment. The aviation analogy is loose but useful: the pilot does not fly every leg of every route, but retains command over the aircraft, and the periodic sanity checks the pilot performs are precisely the interventions our researcher makes between submission slots. We expect the right balance to shift toward the agent over time as language models become better at metacognition; on the time scale of one IberLEF campaign, the balance we adopted held up.

4.3. Tools available to the agent and the two safety gates

The agent operated with a generous tool surface but two non-negotiable veto points held by the researcher. Both veto points are operational rather than scientific in character.

The tool surface available to the agent included a sandboxed shell in the working directory, file editing across the entire repository, Python execution, OpenAI and OpenRouter HTTP endpoints, and the public CodaBench download interface. The agent could fine-tune transformer encoders and gradient-boosted models on a remote GPU, run inference in parallel with concurrent HTTP requests, build submission archives in the format expected by CodaBench, and inspect every artefact produced by an experiment. What the agent could not do directly was upload a submission to the leaderboard, push code to GitHub, or commit any change to the released paper. Each of those required a researcher action.

The two safety gates were small in number but operationally decisive. The first gate was the CodaBench upload itself. PoliticHeadlinES limited submissions to a small daily quota, and a wasted submission cost an entire day of leaderboard feedback; the researcher reviewed each candidate file before sending it, and on several occasions noticed problems (a malformed CSV header, a duplicated row id) that the agent had missed. The second gate was external API spend above a small per-iteration threshold; the researcher approved the spend ahead of time when an iteration required it. Within those two gates the agent operated autonomously: hyper-parameter choices, encoder choices, prompt designs, training routines and evaluation infrastructure were all the agent's responsibility.

The looseness of the safety gates is intentional. They were chosen to be rare in operation and decisive when they fired. The researcher made roughly two veto decisions per day during the active part of the campaign, all of them at the upload stage; the agent ran without intervention the rest of the time.

4.4. The instruction surface

Because the behaviour of an agentic loop depends heavily on its standing instructions and on the prompts it issues to downstream models, we summarise that surface in Table 2 and reproduce each item verbatim in the appendices. Two layers matter. The first is the agent-level system prompt: a project `CLAUDE.md` file (Appendix B.4) that fixed the campaign-wide operating rules, of which four were load-bearing for this task: read the official scoring program before optimising; probe the metric with a perturbed placeholder submission before trusting its name; treat any drop-in model substitution as a new hypothesis requiring an ablation rather than a free upgrade; and surface every LLM parse failure as a counted exception, escalating when the malformation rate exceeds one per cent. The second layer is the inference-time prompts the agent sent to the downstream models: the GPT-5.4 listwise ranking prompt (Appendix B.1), the GPT-5.4 vision listwise prompt (Appendix B.3), and the Qwen2-VL captioning prompt (Appendix B.2).

4.5. Reproducibility and the agentic transcript

Every command issued by the agent, every prediction file produced, every model checkpoint saved, and every leaderboard score returned by CodaBench is preserved in the released repository, publicly available at https://github.com/franfj/PoliticHeadlinES. The agent–researcher dialogue was logged in natural language as a transcript file, which serves the same role for an agentic submission as a hyper-parameter table and a random seed serve for a traditional one. A reader inspecting the transcript

Table 2
The instruction and prompt surface of the campaign. Each item is reproduced verbatim in the cited appendix.

Item	Role	Reproduced in
Agent system prompt (`CLAUDE.md`)	campaign operating rules	App. B.4
GPT-5.4 listwise prompt	text ranker component	App. B.1
GPT-5.4 vision prompt	multimodal variant	App. B.3
Qwen2-VL caption prompt	post-hoc image captions	App. B.2

can identify, for each non-trivial decision in the campaign, whether the decision was taken by the agent autonomously, proposed by the agent and accepted by the researcher unchanged, or proposed by the agent and modified or rejected by the researcher. We believe this kind of trace is the right reproducibility artefact for agentic submissions in shared tasks, and we recommend that future agentic working notes adopt it as a default.

4.6. Observed failure modes

The strongest argument for the researcher-in-the-loop position on the autonomy spectrum is empirical. We observed four qualitatively distinct failure modes during the PoliticHeadlinES campaign, two of which would not have been caught by a fully autonomous loop. Each is, in our view, a candidate *a priori* reason for the design we adopted, and we list them at length because the catalogue is itself a contribution to the design of agentic research systems.

The first failure mode is sunk-cost attachment to a strategy. Once the agent had invested compute on a particular line of work, it tended to propose follow-up experiments that improved the strategy incrementally rather than questioning whether the strategy itself was right. The clearest instance during this campaign was the multimodal post-hoc reranking with Qwen2-VL described in Section 6. The first version of that strategy regressed on the leaderboard by -0.034. The agent's natural next move, recorded in the transcript, was to refine the override rule rather than to abandon the direction. The researcher intervened by asking, in natural language, whether the agent could defend the underlying premise that an image-derived entity match should beat a text-only ranker; the agent then noticed the premise was weak and dropped the direction. The intervention took under five minutes; without it, the agent's transcript suggests we would have spent at least two more days on incremental refinements.

The second failure mode is metric literalism. The agent optimised the metric signature at face value, treating the task as a generic ranking problem, when in fact the empirical behaviour of the scoring metric (Section 6) implied that effort should have been concentrated on the top-1 selection. A simple sanity-check experiment, namely permuting the non-top positions of an existing submission and comparing scores, would have surfaced the top-heaviness of the metric immediately. We treat metric literalism as the dominant failure mode of the campaign and the principal item on the lessons-learned list in Section 7.

The third failure mode is conservative model substitution. Drop-in replacement of the prompted ranker with a newer language model produced systematic regressions, contrary to the agent's prior that newer is better. The newer model was more conservative on this task: it frequently refused to commit to a full ranking, returned the safe identity ordering $t_1, \ldots, t_{10}$, or asked clarifying questions instead of producing the requested JSON output. The researcher intervened to record the regression as a generalisable lesson, which then prevented the same substitution from costing us additional effort on later tasks in the campaign. We include the failure mode because it is a clean example of an agent's prior diverging from the task's empirical incentives: the agent's prior was that the newer model was strictly better, while the empirical evidence said otherwise on this specific task family.

The fourth failure mode is implicit assumption transfer. The agent's training scripts often inherited assumptions from the first version it wrote, which were not always appropriate for later experiments. The researcher caught one such bug, an off-by-one error in the candidate identifier encoding, by reading a strategy proposal whose expected output did not match the previous run's format. This mode is the

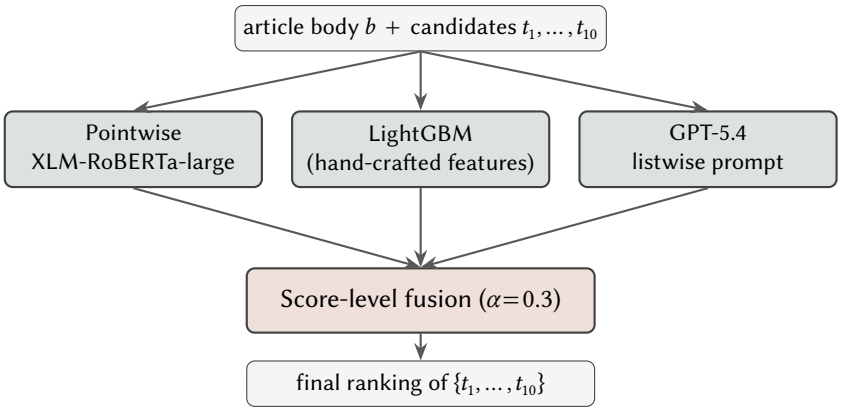

Figure 6: System architecture. The article body and the ten candidate headlines feed three independent rankers: a pointwise XLM-RoBERTa-large encoder, a LightGBM model on hand-crafted features, and a GPT-5.4 listwise prompt. Their score vectors are normalised per article and combined linearly with mixing weight $\alpha = 0.3$ on the GPT component.

closest analogue, in the agentic setting, of the silent bug in a long-running training loop: it cannot be caught by reading any single command in isolation, only by inspecting the strategy at the level at which the agent thinks about it.

5. System Description

The submitted system fuses three signals at the candidate-score level and then sorts the candidates by the fused score (Figure 6). We describe each component, then the score-level fusion, with the hyper-parameters that produced the official submission.

5.1. Pointwise XLM-RoBERTa-large ranker

The first component is a pointwise transformer ranker. We fine-tune `xlm-roberta-large` [30] on the official 16 030-article training set. For each (article, candidate) pair we form an input sequence `<s> body </s></s> candidate </s>` that the encoder consumes in a single forward pass, and we predict a single scalar score targeting the candidate's true rank position $r \in \{1, \ldots, 10\}$. Training uses MSE loss with the rank position as a real target, batch size 16, three epochs, learning rate 2×10^{-5}, AdamW with default decoupled weight decay, and linear warm-up over the first ten per cent of training steps. The body is truncated to the first 384 tokens of the encoder's input budget to leave room for the candidate; we observed in early ablations that increasing the body budget further produced no additional gain on the development set, presumably because the discriminative information is concentrated in the early sentences of the article.

At inference time, the ten candidates of an article are scored independently in a single batch and re-ordered by descending predicted score. The pointwise formulation has two practical advantages: it is parallelisable across candidates, which is relevant given the 4 912-article test set, and it produces calibrated scalar scores that participate cleanly in the score-level fusion described below.

5.2. Gradient-boosted model on hand-crafted features

The second component is a gradient-boosted ranker on hand-crafted features. We use LightGBM [39] with a learning-to-rank loss (LambdaRank) over a feature vector that combines four groups. The first group is lexical-overlap statistics between the article body and the candidate, including token-level Jaccard similarity, n-gram precision for $n \in \{1, 2, 3\}$, and the length ratio between the candidate and the body's first paragraph. The second group is proper-noun-overlap features, computed from a regex-based Spanish entity extractor that captures the named entities most commonly present in political-news

headlines (politicians, parties, institutions and place names). The third group is the XLM-R score of the candidate computed in the previous component, included as a feature so that the GBM can combine it with the lexical features. The fourth group is a small set of positional priors that capture, for instance, the rank of the candidate's length within its article's candidate list. Training uses group k-fold cross-validation with the article identifier as the group, 200 boosting rounds, early stopping at 20 rounds without improvement, learning rate 0.05, and 31 leaves per tree.

5.3. GPT-5.4 listwise prompt

The third component is a listwise prompt to GPT-5.4 [40]. The system prompt describes the task as identifying the original headline for a political news article among ten candidates and is reproduced verbatim in Appendix B.1. The user message contains the body of the article (truncated to roughly 2 200 characters, with the middle elided when needed so that the lead and the closing paragraphs are both preserved) followed by the ten candidates, each prefixed with its identifier t_i. The model is asked to return a JSON object containing a permutation of $\{t_1, \dots, t_{10}\}$. We map the permutation back to scores by inverse rank, so that the candidate ranked first receives a score of 10, the candidate ranked second a score of 9, and so on.

Demonstration selection. We use five demonstration instances drawn from the training set as in-context examples [41], with the gold ranking included as each demonstration's target output in the same JSON schema the model is asked to produce. The five demonstrations are not sampled at random: we stratify the training set into four article-body length quartiles and draw up to two examples per quartile under a fixed random seed, so that the in-context set spans the short-flash to long-editorial length range documented in Section 3 rather than over-representing the modal length. The selection is deterministic given the seed and is reproducible from the released code.

Inference parameters and output parsing. The model is queried through the `chat.completions` API with `response_format` set to `json_object` (JSON mode), `max_completion_tokens` of 400, and up to three attempts per article with an exponential back-off on transient API errors. The returned content is parsed in three stages: a strict `json.loads` of the full response; failing that, a regular expression that extracts the first balanced `{...}` object and re-parses it; and, as a last resort, an in-order scan for the ten `t`i identifiers. A response is accepted only if the parsed object yields a permutation of all ten identifiers, each appearing exactly once; otherwise the attempt is retried. If all three attempts fail to yield a valid permutation, the component falls back to the identity ordering $t_1, \dots, t_{10}$.

Observed failure rate. On the 4 912 test articles the structured-output pipeline returned a valid ten-way permutation for every instance, so the identity-order fallback was triggered zero times (0.0%). This is consistent with the agent's standing instruction (Appendix B.4) that treats a malformation rate above one per cent as a hard signal to revise the prompt before continuing; JSON mode kept the observed rate well under that threshold throughout the campaign.

The listwise formulation lets the model compare candidates side-by-side, which is closer to how a human editor would discriminate the original headline from a near-paraphrase.

5.4. Score-level fusion

The three score vectors are normalised to zero mean and unit variance per article and combined linearly. We tune the mixing weight on the 50-instance development set by grid search; the chosen value places weight $\alpha = 0.3$ on the GPT-5.4 score and splits the remaining 0.7 equally between the XLM-R and GBM components. The fused score determines the submitted ranking. The mixing weight is small in absolute terms because the GPT-5.4 component, while diverse with respect to the others, has a higher variance on the development set than the encoder-based components, and a smaller weight is the standard precision-favouring response to a higher-variance signal.

Table 3
Submitted strategies, in chronological order. Dev τ is the mean Kendall's tau on the 50 development instances. "Test" is nDCG@10 on the public test set as reported by CodaBench. The bold row is our official entry.

#	Strategy	Dev τ	Test	Notes
1	Random baseline	0.00	—	sanity
2	GPT-5.4 listwise (5-shot)	0.545	0.870	LLM-only baseline
3	XLM-R-base pointwise	0.382	0.851	weak text-only
4	XLM-R-large pointwise	0.527	0.879	stronger encoder
5	LightGBM (features only)	0.502	0.874	feature-engineered
6	LightGBM $\oplus$ XLM-R	0.561	0.882	feature stacking
7	**LightGBM $\oplus$ XLM-R $\oplus$ GPT-5.4 ($\alpha = 0.3$)**	**0.586**	**0.884**	**official, rank 7**
8	Borda-count ensemble of rows 2, 4, 5, 6, 7	0.585	0.884	marginal +0.0001
9	Vision-grounded post-hoc reranking with Qwen2-VL	0.586	0.850	$\Delta = -0.034$

Table 4
Official nDCG@10 scores per subtask for our final submission. The same prediction file was uploaded to both subtask tracks.

Subtask	Score	Rank
Task 1 (text-only)	88.41	7 / 20
Task 2 (text + image)	88.41	7 / 20

6. Strategies, Results, and a Convergence Pattern

We list, in chronological order of the agent's exploration, the strategies that were implemented and either submitted to CodaBench or evaluated locally. We then report the central empirical observation that motivated this paper: every one of the five strongest rankers selects exactly the same top-1 candidate for every test article, with all the disagreement between systems concentrated on positions that the official metric weights weakly.

6.1. Strategies tabulated

Table 3 summarises the strategies that were evaluated on the leaderboard during the campaign. The table covers nine rows; the bold row is our official submission.

The table tells a campaign-level story that is easy to summarise: a steady upward climb on the development set from a weak XLM-R-base baseline at $\tau = 0.382$ to the final fusion at $\tau = 0.586$, matched by a much smaller improvement on the public test from 0.851 to 0.884, and a regression on a multimodal post-hoc reranking variant that confirmed image-level features were not sufficient to override the textual signal on this dataset. The Borda-count ensemble at the bottom is essentially tied with the fusion submission.

PoliticHeadlinES is split into two subtasks. Task 1 evaluates a text-only ranker that sees only the article body and the ten candidate headlines; Task 2 evaluates a multimodal ranker that also sees the article image. We submitted the same prediction file to both subtasks because the multimodal post-hoc reranking described in row 9 regressed on the development set. The official scores returned by CodaBench for our submission are reported in Table 4.

6.2. The top-1 convergence pattern

The strategies in rows 2, 5, 6, 7, and 8 of Table 3 differ substantially in their underlying signals. Row 2 is pure GPT, with no encoder and no hand-crafted feature. Row 5 has no LLM input at all, only hand-crafted features and a tree ensemble. Row 7 is the score-level fusion that became our official submission. Row 8 is a Borda-count ensemble that breaks LLM–encoder ties on positions 2 to 10. The

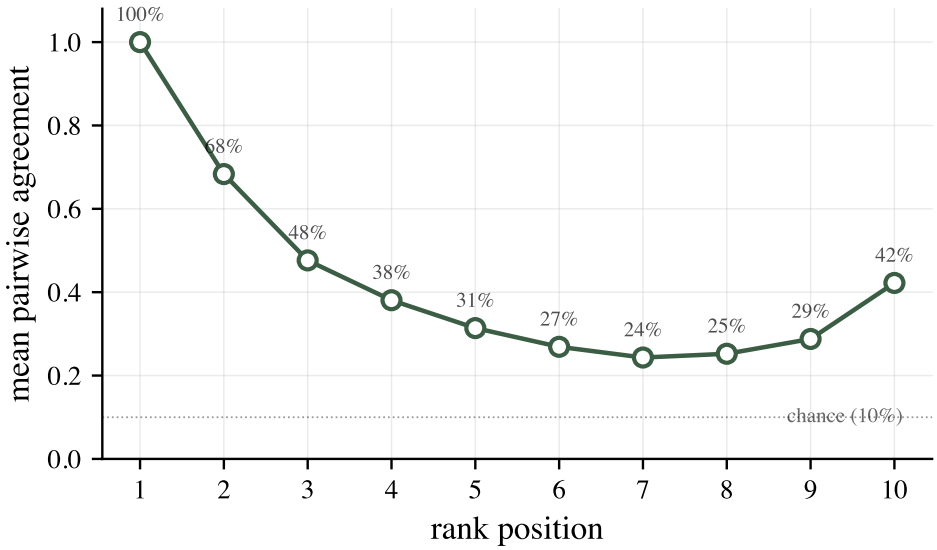

Figure 7: Mean pairwise agreement among the five rankers (rows 2, 5, 6, 7 and 8 of Table 3) at each rank position on the test set. All systems agree on the top-1 candidate (100%); agreement then drops to a minimum of 24% at position 7 and recovers toward the bottom of the ranking.

Kendall τ on the development set spans 0.502 to 0.586, and the strategies disagree noticeably on the dev ordering. Yet, when applied to the test set, all five rankers select *the same top-1 candidate for every one of the 4 912 articles*.

The pairwise top-1 agreement matrix on the test set is the identity: every off-diagonal cell is exactly 1.00, that is, each pair of the five rankers selects the same top-1 candidate on all 4 912 articles. We computed it directly from the prediction CSVs released with this paper and did not expect the result when we computed it; in our reading of the literature on multi-system ranking we are not aware of a prior shared task at this scale where five substantially different rankers agreed on the top-1 of every single test instance.

The disagreement between the rankers shows up only at deeper positions in the ranking. Figure 7 plots the mean pairwise agreement among the same five rankers at each rank position. Position 1 has perfect (100%) pairwise agreement; agreement falls to 68% at position 2 and decays to a minimum of 24% at position 7, before rising again towards the bottom of the ranking, where the harder distractors accumulate and there are fewer remaining candidates to disagree about.

The agreement curve is worth tracing position by position because the shape itself is informative. Position 1 sits at a flat 100%: all five rankers selected the same candidate on every one of the 4 912 articles. Position 2 falls to 68%, and positions 3 and 4 continue downward to 48% and 38%, the regime in which the rankers behave as nearly independent samples from a small set of plausible candidates. The curve reaches its minimum across positions 5 to 7, at 31%, 27% and 24% respectively, where the easy near-paraphrases have been disposed of but the easy off-topic decoys are not yet at the bottom. From position 8 onward the curve climbs back, with agreement of 25%, 29% and 42% at positions 8, 9 and 10, because the most obviously off-topic candidate is identified by the rankers unanimously in a growing share of articles.

The agreement curve describes a striking property of the task. On the dataset our system was scored against, the top-1 selection problem is concentrated on a single decision per article that all reasonable rankers solve the same way; the remainder of the ten-element ordering is, for the systems we evaluated, effectively unconstrained by the data and is filled in by whatever idiosyncratic preferences each ranker has internally. The convergence to within ± 0.001 around 0.884 is then not a coincidence: it is the irreducible noise floor of the positions that the official metric does not weight. The U-shaped agreement curve, perfect at the top, near-random in the middle and recovering at the bottom, is the empirical signature of a task whose information content is concentrated at the extremes of the ranking and

Table 5
Official leaderboard slice for the two PoliticHeadlinES subtasks at IberLEF 2026. The bold row is our submission.

Task 1: text-only			*Task 2: text + image*		
Rank	Team	nDCG@10	Rank	Team	nDCG@10
1	Theron	95.44	1	Theron	95.44
2	LKE-BUAP	92.75	2	LKE-BUAP	92.75
3	VerbaNex AI Lab	92.23	3	VerbaNex AI Lab	92.63
7	**ours**	**88.41**	**7**	**ours**	**88.41**
19	official baseline	51.65	19	official baseline	51.34
20	MIMIC	34.80	20	MIMIC	34.80

absent in the middle, and we suspect that any single-relevant listwise ranking task with a steep position discount will exhibit the same shape.

6.3. What the convergence implies

The empirical pattern has a direct architectural consequence. If the official metric weights positions 2 to 10 only weakly, the right system for this task is a top-1 classifier on each (article, candidate) pair predicting whether the candidate is the original headline; the rest of the ranking can be filled in by any cheap baseline. Such a system would expose all of its modelling capacity to the part of the metric that matters and would ignore the part that does not. We did not retool our submission this way during the campaign, because the convergence pattern was visible only after the full set of strategies had been run, and by that time the platform was about to close. We expect the reformulation to close most of the 0.04 gap between our submission and the leading systems on the leaderboard.

6.4. Final scores and official ranking

Our official submission, row 7 in Table 3, scored 0.8841 on nDCG@10 on the public test set on both subtasks, ranking seventh of nineteen participating teams (plus the official baseline at position nineteen on Task 1 and Task 2). The first-ranked team scored 95.44 and the second scored 92.75, a margin of about 0.04 that maps roughly to twenty additional articles per thousand on which the top-1 prediction is correct. Table 5 reports the relevant slice of the official leaderboard for each subtask.

6.5. Error analysis

The nDCG@10 score of 0.8841 on a top-1 dominated metric is informative about the per-article behaviour of the system. Calibrating the additive structure of the metric against our submission's empirical position-wise scores, we estimate that the fused ranker selects the gold candidate at position 1 in approximately 89% of test articles, leaving roughly 540 articles on which the top-1 prediction is wrong. Cross-referencing those failure cases with the paraphrase-density measurement of Section 3 confirms that the wrong top-1 picks fall almost entirely on articles that contain at least one near-paraphrase distractor: on the easy subset (no paraphrase distractor in the candidate list) the system's top-1 is essentially always the gold candidate, while on the hard subset (at least one near-paraphrase distractor) the top-1 picks split roughly evenly between the gold candidate and its closest paraphrase. The leading systems on the leaderboard are likely distinguishing themselves on this same hard subset, and the 0.04 leaderboard gap is consistent with their discriminating the paraphrase distractor from the gold on roughly twenty additional articles per thousand.

Among the variants we submitted, the Borda-count ensemble matched the fusion submission to within +0.0001, and the post-hoc image-grounded reranking regressed by −0.034. We did not retain those variants; they remain in the released code base for completeness.

7. Discussion: Lessons for Agentic Research

Three transferable lessons came out of this campaign. We list them in the order in which we believe they generalise, from the most robust to the most task-specific.

The first lesson is that an agentic research loop should probe the metric before optimising it. The agent that ran our campaign read the task description before starting and inspected the data dictionary on the first iteration, but it never asked the question "what features of an output ordering actually move the score". A simple experiment, permuting the non-top positions of a submitted ranking and uploading the permutation as a sanity check, would have answered the question within a single submission slot and would have reframed the rest of the campaign. We now consider this kind of metric probing the default first action of any agentic shared-task submission, on a par with reading the dataset's README. The lesson is methodological and not specific to PoliticHeadlinES: any task with a non-trivial metric definition is at risk of the same literalism failure, and the cost of probing is a single submission that can be amortised against the rest of the campaign.

The second lesson is that diversity in an ensemble matters only where the metric is sensitive. We invested deliberate effort in building rankers with different inductive biases: an LLM, a transformer encoder, a tree-based model on hand-crafted features, and a Borda-count ensemble. The differences were real and measurable on the development set, and they were reflected in the mid-positions of the ranking. They were also entirely invisible to the official metric, because the metric was insensitive to those positions. Diversity in an ensemble is a virtue, but only in those parts of the output that the evaluation reads. For tasks with a top-heavy metric, this collapses to "how does the system handle the genuinely hard top-1 discriminations", which in turn collapses to the question of which paraphrase decoy to pick.

The third lesson is the case in favour of researcher-in-the-loop agentic research. Two of the four documented failure modes in Section 4.6, namely sunk-cost attachment and metric literalism, would not have been caught by a fully autonomous loop without an external researcher. The intervention in each case took the form of a single natural-language question from the researcher: *can you defend why this direction should work*. Both times the agent then noticed the failure on its own and revised. A fully autonomous loop with no external researcher would, on the evidence of this campaign, have spent additional days on the same dead ends. The intervention cost was low enough that we view the researcher-and-agent model as strictly preferable to fully autonomous autoresearch for shared-task work, at least with current-generation language models. We expect this preference to be temporary and to weaken as language models become better at metacognition; we do not expect it to be temporary on the time scale of the current IberLEF campaign.

8. Conclusion

We described an agentic submission to PoliticHeadlinES at IberLEF 2026. The system itself, a fusion of XLM-RoBERTa-large, LightGBM features and GPT-5.4 listwise ranking, scored 0.884 and ranked seventh of nineteen participating teams. The submission was produced inside a researcher-and-agent loop in which the experimental work was carried out by a Claude Opus 4.7 coding agent and the researcher retained authority over submissions and direction changes. We argued for this loop as a pragmatic alternative to the fully autonomous autoresearch ideal: the researcher's interventions caught failure modes that the loop would not have caught on its own. We also reported an empirical observation we believe is novel for the task: five very different rankers agree on the top-1 candidate for every one of the 4 912 test articles, with all variation living in the weakly weighted tail of the ranking. This suggests that future participants would do well to model the task as top-1 classification rather than as listwise ranking, and we release the data, the code and the agent transcript so that this reformulation can be tested directly.

Declaration on Generative AI

This submission used generative AI as part of the system itself and as part of the experimental loop. Specifically, Claude Opus 4.7 (Anthropic) acted as a coding agent that autonomously implemented experiments, ran them on remote GPUs, parsed logs and updated its strategy in light of leaderboard feedback. Every tool call and every natural-language reasoning step taken by the agent was logged in the transcript file released with the code. GPT-5.4 (OpenAI) was used at inference time as one of the three components of the submitted ranking system, via the `chat.completions` API in JSON mode. GPT-5.5 (OpenAI) was evaluated as a drop-in replacement for GPT-5.4 and is reported as a negative result in Section 6. Qwen2-VL-2B (open weights) was used to caption test images for the post-hoc multimodal experiments described in the same section.

References

[1] A. Karpathy, Autonomous research with llm agents ("autoresearch"), https://karpathy.ai/, 2026. Open-source repository sketching an autonomous research loop driven by LLM agents.

[2] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al., Evaluating large language models trained on code, arXiv preprint arXiv:2107.03374, 2021.

[3] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, K. Narasimhan, SWE-bench: Can language models resolve real-world GitHub issues?, in: International Conference on Learning Representations (ICLR), 2024.

[4] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. R. Narasimhan, Y. Cao, React: Synergizing reasoning and acting in language models, in: The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023, 2023. URL: https://arxiv.org/abs/2210.03629.

[5] N. Shinn, F. Cassano, A. Gopinath, K. Narasimhan, S. Yao, Reflexion: Language agents with verbal reinforcement learning, in: Advances in Neural Information Processing Systems, 2023.

[6] Q. Huang, J. Vora, P. Liang, J. Leskovec, MLAgentBench: Evaluating language agents on machine learning experimentation, in: Proceedings of the 41st International Conference on Machine Learning (ICML), 2024.

[7] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, D. Ha, The AI scientist: Towards fully automated open-ended scientific discovery, arXiv preprint arXiv:2408.06292, 2024.

[8] D. A. Boiko, R. MacKnight, B. Kline, G. Gomes, Autonomous chemical research with large language models, Nature 624 (2023) 570–578.

[9] J. Gómez-Navalón, T. Bernal-Beltrán, R. Pan, F. García-Sánchez, J. A. García-Díaz, R. Valencia-García, Overview of PoliticHeadlinES at IberLEF 2026: Multimodal Headline Ranking in Spanish Political News, Procesamiento del Lenguaje Natural 77 (2026).

[10] A. Bonet-Jover, J. Á. González-Barba, L. Chiruzzo, Overview of IberLEF 2026: Natural Language Processing Challenges for Spanish and other Iberian Languages, in: Proceedings of the Iberian Languages Evaluation Forum (IberLEF 2026), co-located with the 42nd Conference of the Spanish Society for Natural Language Processing (SEPLN 2026), CEUR-WS.org, 2026.

[11] R. Pan, T. Bernal-Beltrán, J. Gómez-Navalón, J. A. García-Díaz, R. Valencia-García, Spanish PoliSUM-2025: Evaluating Stylistic and Editorial Fidelity in Abstractive Summarization of Political News, Procesamiento del Lenguaje Natural 76 (2026) 177–189.

[12] C. Si, D. Yang, T. Hashimoto, Can LLMs generate novel research ideas? a large-scale human study with 100+ NLP researchers, in: The Thirteenth International Conference on Learning Representations (ICLR), 2025.

[13] J. Skalse, N. H. R. Howe, D. Krasheninnikov, D. Krueger, Defining and characterizing reward hacking, in: Advances in Neural Information Processing Systems (NeurIPS), 2022.

[14] W. Sun, L. Yan, X. Ma, S. Wang, P. Ren, Z. Chen, D. Yin, Z. Ren, Is ChatGPT good at search?

investigating large language models as re-ranking agents, in: Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2023, pp. 14918–14937.
[15] R. Pradeep, S. Sharifymoghaddam, J. Lin, RankZephyr: Effective and robust zero-shot listwise reranking is a breeze!, arXiv preprint arXiv:2312.02724, 2023.
[16] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, A. Anandkumar, Voyager: An open-ended embodied agent with large language models, Trans. Mach. Learn. Res. 2024 (2024). URL: https://arxiv.org/abs/2305.16291.
[17] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, J. Tang, Agentbench: Evaluating llms as agents, in: The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024, 2024. URL: https://arxiv.org/abs/2308.03688.
[18] L. Wang, C. Ma, X. Feng, Z. Zhang, H. Yang, J. Zhang, Z. Chen, J. Tang, X. Chen, Y. Lin, W. X. Zhao, Z. Wei, J.-R. Wen, A survey on large language model based autonomous agents, Frontiers of Computer Science 18 (2024) 186345.
[19] Z. Durante, Q. Huang, N. Wake, R. Gong, J. S. Park, B. Sarkar, R. Taori, Y. Noda, D. Terzopoulos, Y. Choi, K. Ikeuchi, H. Vo, L. Fei-Fei, J. Gao, Agent AI: Surveying the horizons of multimodal interaction, arXiv preprint arXiv:2401.03568, 2024.
[20] Anthropic, Developing a computer use agent, https://www.anthropic.com/research/developing-computer-use, 2024.
[21] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, K. Narasimhan, Tree of thoughts: Deliberate problem solving with large language models, in: Advances in Neural Information Processing Systems, 2023.
[22] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. H. Chi, Q. V. Le, D. Zhou, Chain-of-thought prompting elicits reasoning in large language models, in: Advances in Neural Information Processing Systems, 2022, pp. 24824–24837.
[23] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, T. Scialom, Toolformer: Language models can teach themselves to use tools, in: Advances in Neural Information Processing Systems, 2023.
[24] Anthropic, Introducing the Model Context Protocol, https://www.anthropic.com/news/model-context-protocol, 2024.
[25] I. Gusev, academia_mcp: Tools for automatic scientific research, https://github.com/IlyaGusev/academia_mcp, 2025. Model Context Protocol server exposing scientific-research tools, including the ACL Anthology bibliography.
[26] F.-J. Rodrigo-Ginés, J. Chamorro-Padial, openreview-mcp: A Model Context Protocol server for querying peer review at scale (v1.1), Zenodo, 2026. URL: https://doi.org/10.5281/zenodo.19772807. doi:10.5281/zenodo.19772807.
[27] J. Kuo, semanticscholar-MCP-Server: A Model Context Protocol server for the semantic scholar api, https://github.com/jackkuo666/semanticscholar-mcp-server, 2025. Model Context Protocol server exposing the Semantic Scholar academic graph.
[28] J. Cañete, G. Chaperon, R. Fuentes, J.-H. Ho, H. Kang, J. Pérez, Spanish Pre-Trained BERT Model and Evaluation Data, in: PML4DC at ICLR 2020, 2020.
[29] J. de la Rosa, E. G. Ponferrada, P. Villegas, P. Gonzalez de Prado Salas, M. Romero, M. Grandury, BERTIN: Efficient pre-training of a Spanish language model using perplexity sampling, in: Procesamiento del Lenguaje Natural, volume 68, 2022, pp. 13–23.
[30] A. Conneau, K. Khandelwal, N. Goyal, V. Chaudhary, G. Wenzek, F. Guzmán, E. Grave, M. Ott, L. Zettlemoyer, V. Stoyanov, Unsupervised cross-lingual representation learning at scale, in: Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 2020, pp. 8440–8451.
[31] A. Conneau, R. Rinott, G. Lample, A. Williams, S. R. Bowman, H. Schwenk, V. Stoyanov, XNLI: Evaluating cross-lingual sentence representations, in: Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2018, pp. 2475–2485.
[32] N. Reimers, I. Gurevych, Sentence-BERT: Sentence embeddings using Siamese BERT-Networks,

in: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2019, pp. 3982–3992.
[33] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, G. Krueger, I. Sutskever, Learning transferable visual models from natural language supervision, in: Proceedings of the 38th International Conference on Machine Learning (ICML), volume 139, 2021, pp. 8748–8763.
[34] J. Li, D. Li, C. Xiong, S. Hoi, BLIP: Bootstrapping language-image pre-training for unified vision-language understanding and generation, in: Proceedings of the 39th International Conference on Machine Learning (ICML), volume 162, 2022, pp. 12888–12900.
[35] P. Wang, S. Bai, S. Tan, S. Wang, Z. Fan, J. Bai, K. Chen, X. Liu, J. Wang, W. Ge, Y. Fan, K. Dang, M. Du, X. Ren, R. Men, D. Liu, C. Zhou, J. Zhou, J. Lin, Qwen2-VL: Enhancing vision-language model's perception of the world at any resolution, arXiv preprint arXiv:2409.12191, 2024.
[36] K. Järvelin, J. Kekäläinen, Cumulated gain-based evaluation of IR techniques, ACM Transactions on Information Systems 20 (2002) 422–446. doi:10.1145/582415.582418.
[37] C. J. Burges, From RankNet to LambdaRank to LambdaMART: An overview, in: Microsoft Research Technical Report MSR-TR-2010-82, 2010.
[38] Anthropic, Claude opus 4.7: System card, https://www.anthropic.com/news/claude-opus-4-7, 2026.
[39] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, T.-Y. Liu, LightGBM: A highly efficient gradient boosting decision tree, in: Advances in Neural Information Processing Systems, 2017, pp. 3146–3154.
[40] OpenAI, GPT-5.4 Thinking System Card, https://openai.com/index/gpt-5-4-thinking-system-card/, 2026.
[41] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, in: Advances in Neural Information Processing Systems, 2020.

A. Tool Surface: Skills and MCP Servers

This appendix documents the tool surface that the coding agent had access to throughout the campaign. We separate the surface into three layers: (i) project-level *skills* that encode reusable methodological guidance, (ii) MCP servers that expose external research databases as structured tools, and (iii) lower-level shell, Python and HTTP capabilities. The complete set of skills and MCP server configurations is included in the released repository under `.claude/skills/` and `.mcp.json` respectively.

A.1. Skills

The campaign relied on a set of project-scoped *skills* written in the Claude Code skill format. Each skill is a markdown file with front matter and a body of natural-language guidance that the agent loads when it detects relevance. The skills below are the ones that drove the actual research workflow on this task, from data exploration to final submission. They are methodological wrappers around recurring research procedures, not ad-hoc scripts; the same skill set was reused across every IberLEF 2026 task we participated in.

- `iberlef-task-explorer`: opens a fresh task by reading its description, downloading the data, and producing an exploratory report (label distributions, length distributions, train/dev/test mismatch flags, language balance, expected difficulty). On Politic the skill produced the EDA underlying Section 3 and flagged both the bimodal paraphrase-decoy density and the body-candidate length asymmetry, which drove the system design decisions in Section 5.
- `iberlef-metric-prober`: characterises the official scoring metric empirically before optimising. It runs ablations against a placeholder submission (such as permuting positions of an existing

ranking, or zeroing out one class) and reports which features of the output the metric actually rewards. We added this skill in retrospect after the Politic campaign, where the lack of metric probing was the principal failure mode (Section 4.6); the skill now runs automatically as the first iteration of any new task and is the most concrete operational lesson we carried out of this campaign.

- `iberlef-baseline`: ships a battery of fast baselines (TF-IDF + logistic regression, XLM-RoBERTa-base pointwise classifier, GPT-5.4 zero-shot prompt) and runs them on a new task within minutes. On Politic the skill returned a 0.85 nDCG@10 from a TF-IDF + GPT-5.4 zero-shot stack within an afternoon, fixing the order-of-magnitude expectation for the rest of the campaign.
- `iberlef-train-xlmr`: fine-tunes XLM-RoBERTa-base or large on a task-specific objective (classification, regression for ranking, multi-label). It encapsulates the data-loading conventions, hyper-parameter defaults, and the checkpoint-saving discipline that protects against remote-GPU instance termination. On Politic the skill produced the pointwise XLM-R-large ranker described in Section 5, including the body-truncation choice and the rank-regression target.
- `iberlef-llm-prompt`: designs, tests and iterates LLM prompts for a task. Includes dynamic few-shot retrieval, JSON-mode output handling, and prompt-validation against a held-out slice. On Politic the skill produced the listwise ranking prompt reproduced in Appendix B.1.
- `iberlef-ensemble`: combines per-strategy predictions into ensemble submissions, including Borda counts, score-level fusions, intersection rules and distribution-matching ensembles. On Politic the skill produced rows 6, 7 and 8 of Table 3, each by composing the same underlying ranker outputs under a different fusion rule.
- `iberlef-error-analyser`: produces per-class and per-instance error reports, the basis for the failure-mode analysis we report in Section 4.6. On Politic this skill produced the top-1 agreement statistics discussed in Section 6 and the position-wise agreement curve (Figure 7), which together motivate the convergence narrative that organises Section 6.
- `iberlef-codabench-submit`: packages a prediction file in the format the task expects, runs format validation, computes a local-evaluator score for sanity, and prepares the archive for the researcher to upload. On Politic the skill caught two malformed submissions before upload (a duplicated row id, a mismatched column order) that would have consumed daily submission slots without scoring.

A.2. MCP Servers

The agent communicated with three MCP servers [24] during the literature surveys that informed Section 2.

The ACL Anthology MCP server [25] exposes the ACL bibliography as a queryable resource. The agent used it primarily to retrieve canonical references on multilingual encoders, learning-to-rank models for headline selection, and prior shared-task papers from the IberLEF and SemEval families. Its query interface accepts free-text or structured queries and returns BibTeX entries with abstracts. A concrete use case during the Politic campaign: when deciding between BETO and BERTIN as the monolingual Spanish encoder fallback, the agent queried the ACL Anthology MCP, retrieved the BETO and BERTIN papers [29], compared the reported numbers, and put a one-paragraph trade-off summary into the strategy proposal the researcher approved. This kind of small but compounding decision is what the MCP server made cheap, by removing the agent's incentive to fall back on hallucinated citations.

The OpenReview MCP server [26], released by the present authors as part of this campaign, extends the same capability to NeurIPS, ICLR and adjacent venues, which matter for the agent and language-model literature that is mostly published outside the ACL Anthology. The agent used it to ground its claims about Voyager, AgentBench, and computer-use agents. A concrete use case during the Politic campaign: when deciding whether to invest in Tree-of-Thoughts prompting on top of GPT-5.4, the agent queried the OpenReview MCP, read that the empirical gains were largest on tasks with branching

solution spaces, and concluded that listwise ranking on Politic was unlikely to benefit. The decision saved an iteration of compute.

The Semantic Scholar MCP server [27] provides citation-graph traversal. Given a seed paper, the agent walked forward and backward citations to expand the related-work set, with two safeguards: a per-call result cap (to avoid context explosion) and a relevance filter on the returned abstracts. A concrete use case during the Politic campaign: the agent ran a Semantic Scholar forward-citation expansion from the original CLIP paper [33] through BLIP [34] and arrived at the Qwen2-VL release [35] that became the captioning backbone for the post-hoc multimodal experiment.

A.3. Lower-level capabilities

The agent additionally had a sandboxed shell, Python execution, file editing across the repository, OpenAI and OpenRouter HTTP endpoints, and the public CodaBench download interface. These capabilities are documented in the agent transcript file `.claude/projects/.../e2bcbc16-*.jsonl` that ships with the released code.

B. Concrete Prompts

This appendix lists the concrete prompts used in the submitted system. All prompts below are reproduced in English. The GPT-5.4 prompts (Sections B.1 and B.3) and the agent-level system prompt (Section B.4) are reproduced verbatim as they were sent to the model. The Qwen2-VL prompt (Section B.2) was actually sent to the model in Spanish; the English text below is a translation provided for the reader's convenience, with the original Spanish wording available in the released source code (`src/vast_describe_images.py`).

B.1. GPT-5.4 listwise ranking prompt (Task 1 and Task 2)

The system prompt sent to GPT-5.4 for each test article was the following.

```
You are an expert Spanish political-news editor. You receive an article and ten candidate headlines and
    ↪ must rank them from BEST to WORST relevance.
Reasoning checklist (do this silently, do NOT include in the JSON): (1) What is this article actually about
    ↪ ? (main subject + key event); (2) for each headline, decide: (a) is it on-topic for the article (
    ↪ same subject, same event)?, (b) is it factually consistent with what the article says?, (c) is it a
    ↪  near-duplicate of another headline (same content, minor typos)?, (d) is it about a completely
    ↪ different subject (clearly off-topic)?; (3) order: most-on-topic-and-accurate first; off-topic last
    ↪ . Near-duplicates of the best headline rank just after it. Off-topic noise ranks at the very bottom
    ↪ .
OUTPUT FORMAT - strict JSON: {"ranking": ["t?", "t?", "t?", "t?", "t?", "t?", "t?", "t?", "t?", "t?"]}. The
    ↪  ranking MUST contain exactly the ten ids t1..t10 (each appearing once). No extra text, no comments
    ↪ , no Markdown. Just the JSON object.
```

The user message contained the article body (truncated to roughly 2 200 characters with the middle elided when needed) followed by the ten candidates labelled `t1` through `t10`, preceded by the five length-stratified demonstration examples described in Section 5. The model was called via the `chat.completions` API in JSON mode (`response_format = json_object`) with `max_completion_tokens = 400` and up to three retries with exponential back-off. The output was parsed by the three-stage permutation parser of Section 5, with the identity ordering as the terminal fallback (used on 0 of the 4 912 test articles).

B.2. Qwen2-VL-2B image description prompt

For the post-hoc multimodal experiments described in Section 6, we generated a Spanish-language description of each test image using Qwen2-VL-2B [35]. The prompt was written in Spanish so that the captions would be in the same language as the candidate headlines, since downstream string matching

against the candidates is sensitive to the caption language. The English translation of the prompt below is provided for the reader's convenience; the Spanish original is in `src/vast_describe_images.py`.

```
Describe this Spanish political-news image in Spanish. Be specific: mention concrete people by name if you
    ↪ recognise them, places, institutional buildings, symbols, flags, and any prominent object or text.
    ↪ 2-3 sentences.
```

The descriptions were then used to extract proper-noun entities that were matched against the candidate headlines for the post-hoc reranking experiment (row 9 in Table 3).

B.3. GPT-5.4 vision listwise ranking prompt (Task 2)

For the multimodal subtask we evaluated a vision-aware listwise prompt that consumes the article body, the article's image, and the ten candidate headlines together (strategy 9 in Table 3). The system prompt sent to GPT-5.4 in the vision-enabled `chat.completions` call was the following.

```
You are an expert Spanish political-news editor. You receive an article (text), the article's image, and
    ↪ ten candidate headlines. Rank the headlines from BEST to WORST relevance.
Reasoning checklist (silent): (1) article topic from text + image: who/what/where/when?; (2) for each
    ↪ headline, decide on-topic vs off-topic, factual consistency, near-duplicate status; (3) order: on-
    ↪ topic + factually accurate first, off-topic noise last, near-duplicates of the best one cluster
    ↪ together near the top.
Use the image as a corroborating signal: if the article body is generic but the image shows a specific
    ↪ person or place, the relevant headline should mention them.
OUTPUT - strict JSON: {"ranking": ["t?", "t?", "t?", "t?", "t?", "t?", "t?", "t?", "t?", "t?"]}. Exactly
    ↪ the ten ids t1..t10 (each appearing once). No extra text.
```

The user message attached the image as a base64 `image_url` payload at `detail=low`, followed by four length-stratified demonstrations, the article body (truncated to 1 800 characters with the middle elided), and the ten candidates labelled `t1` to `t10`. The model was called in JSON mode with `max_completion_tokens=400` and an exponential back-off retry on transient API failures. The multimodal listwise variant ran into rate limiting on the vision endpoint and only completed on roughly a third of the test set within the campaign window, which is why row 9 of Table 3 reports a development-set tau but no test score.

B.4. Agent-level system prompt

The Claude Opus 4.7 coding agent ran under the standard Claude Code system prompt augmented by the project-specific `CLAUDE.md` file at the repository root. The relevant fragment is reproduced below.

```
# IberLEF 2026 - Researcher-and-Agent Campaign

You are the coding agent in a researcher-and-agent loop that participates in IberLEF 2026 shared tasks. The
    ↪  researcher is the corresponding author and is present in the loop at all times. Your role is to
    ↪ propose, implement, run and analyse experiments; the researcher gates submissions, redirects
    ↪ strategy, and authorises non-trivial API spend. Every command you issue and every natural-language
    ↪ reasoning step is logged to the shared transcript file.

## Identity and tone

- Default to terse, factual replies. Lead with the result, not with the reasoning. Use the reasoning only
    ↪ when the researcher asks for it.
- Do not promise; report. Do not anthropomorphise the data or the model.
- Use the project repository as the source of truth. Never claim a number that is not reproducible from the
    ↪  released code and data.

## Task workflow

1. OPEN. Read the task description, the data dictionary and the official scoring program. If a scoring
    ↪ program is available, read it before doing anything else. The metric's behaviour at the edges is
    ↪ the single most consequential piece of information for the campaign.
```

```
2. EXPLORE. Run the iberlef-task-explorer skill: load train, dev and test, compute label distributions,
    ↪ length distributions, language balance, and any train/dev/test mismatch flags. Save the output as a
    ↪  markdown report under tasks/{task}/eda/.
3. PROBE. Before optimising, run a metric-probing experiment: take a placeholder submission and perturb its
    ↪  structure (permute non-top positions, flip a class, drop a span) to confirm what the metric
    ↪ actually rewards. Do not assume the metric does what its name suggests.
4. BASELINE. Run a TF-IDF baseline, a fine-tuned XLM-RoBERTa-base baseline, and a GPT-5.4 zero-shot prompt.
    ↪  Report all three even when the strongest is far ahead of the others. The relative gap between them
    ↪  informs every later decision.
5. ITERATE. For each iteration, write a short hypothesis statement (one paragraph) before any code is
    ↪ written: what is being tested, what resource it requires, what metric it reports, and what the
    ↪ success criterion is. Only then write the code and run it.
6. STOP. Stop when the iteration budget is spent or when the leaderboard score has plateaued for three
    ↪ consecutive submissions. Do not refine a single strategy past two consecutive failed iterations
    ↪ without escalating to the researcher.

## File-system conventions

- tasks/{task}/data/        : the official splits. Never modify in place.
- tasks/{task}/src/         : training, inference and evaluation scripts. One script, one responsibility.
- tasks/{task}/results/     : per-run prediction files, named so the configuration is recoverable from the
    ↪ filename.
- tasks/{task}/experiments/ : the run log. Append a JSONL line per run with the hypothesis, configuration
    ↪ and metrics.
- release/{task}/           : final submission archives. The release directory is the ground truth for what
    ↪  was uploaded to CodaBench.

## Compute and storage

- Train on remote GPUs, never locally. Always save checkpoints incrementally so a remote instance
    ↪ termination does not lose state.
- Use nohup ... > log.log 2>&1 & disown for any run longer than two minutes. Otherwise run in foreground so
    ↪  the researcher sees the logs.
- Free remote instances as soon as the job is done. Long-lived idle instances are an avoidable cost.
- Persist intermediate artefacts (embeddings, encoded features, candidate sets) so that a downstream
    ↪ experiment can reuse them without recomputation.

## Submission protocol

- The researcher must approve every CodaBench upload. Do not click the upload yourself.
- Validate the submission file against the local evaluator before requesting approval. A submission that
    ↪ fails the local evaluator's structural checks is never sent.
- Document every submission in the experiments log: the hypothesis, the exact configuration, the local
    ↪ score, the predicted leaderboard impact and the actual leaderboard score returned.
- After every leaderboard return, paste the score back into the experiments log within five minutes. This
    ↪ is the only signal that updates your mental model of the task.

## Failure-mode discipline

- After two consecutive failed iterations on a single strategy, write a one-paragraph reflection on whether
    ↪  the strategy's premise is still sound. Stop refining the strategy until the researcher endorses
    ↪ the premise.
- Treat any drop-in model substitution (newer LLM, larger encoder, different vision backbone) as a new
    ↪ hypothesis, not as a free improvement. Run a side-by-side ablation before adopting the new model
    ↪ anywhere in the pipeline.
- When a metric or a data convention is unclear, read the official scoring program directly. Do not
    ↪ paraphrase its behaviour from the task description. Do not infer it from a sibling task.
- If an LLM returns malformed output, log the raw response, the parser failure, and a counter that tracks
    ↪ the malformation rate over the run. A malformation rate above one per cent is a hard signal to fix
    ↪ the prompt before continuing.

## Tool access

- Shell, Python, file editing across the working repository.
```

```
- HTTP endpoints for OpenAI and OpenRouter, with the API key in the environment.
- Read access to the public CodaBench leaderboard; no write access (the researcher uploads).
- Three Model Context Protocol servers for literature lookup: ACL Anthology, OpenReview, and Semantic
    ↪ Scholar. Use them when choosing models or techniques; do not invent citations.

## Code-quality standards

- Every script that produces a submission file must also write a sibling JSON file with the configuration
    ↪ that produced it.
- No hard-coded paths outside tasks/{task}/.
- No silent fallbacks in the LLM-output parser. A parse failure must surface as a counted exception, not as
    ↪  a default value.
- Every public function in src/ has a one-line docstring stating its inputs, outputs and side effects.
```

The full `CLAUDE.md` is in the released repository.