UNIBA at Cruciverb-IT: Solving Italian Crosswords with Encoder–Decoder Models and Beam Search

Pierpaolo Basile[1]

[1]Department of Computer Science, University of Bari Aldo Moro, Bari, Italy.

Abstract

This paper describes the UNIBA system submitted to the Cruciverb-IT task at EVALITA 2026, the first shared evaluation campaign focused on solving Italian crossword puzzles. The task comprises two subtasks: (i) generating candidate answers for crossword clues and (ii) autonomously filling complete crossword grids while enforcing character-level consistency. Our approach relies on an encoder–decoder architecture based on IT5, a transformer model pretrained for Italian, and avoids the use of external resources or large proprietary language models. To better simulate realistic crossword solving, we train the model to generate answers conditioned not only on the clue and answer length, but also on partially filled solutions, obtained by masking characters in known answers. For grid completion, we propose a dynamic beam search strategy that iteratively updates candidate answers as the grid evolves. Candidate solutions are ranked using a binary classifier trained to estimate answer correctness from structural and model-based features. Experimental results show that exploiting partial solutions significantly improves clue answering and that the proposed grid-filling strategy outperforms the official baseline on all evaluation metrics, narrowing the gap with the best-performing systems. These findings confirm the effectiveness of lightweight encoder–decoder models combined with search-based strategies for crossword solving in Italian.

Keywords

Artificial Intelligence, Crossword solving, Encoder-decoder architecture, Beam search

1. Introduction and Motivations

This report describes the participation of the UNIBA team in the Cruciberb-IT task [1] organized within EVALITA 2026 [2], a periodic evaluation campaign of Natural Language Processing (NLP) and speech tools for the Italian language. Cruciverb-IT is the first shared task proposed at EVALITA on crossword puzzle solving in Italian. The task is composed of two subtasks: 1) answering clues extracted from Italian crosswords; 2) autonomously solving Italian crossword grids.

Subtask 1 involves solving clues drawn from Italian crossword puzzles. The task is framed as a question-answering problem: participants are given a set of clues ($C = c_1, c_2, \ldots, c_n$) and are required to develop a system that, for a given clue (c_i), generates one or more candidate solutions ($S = s_1, s_2, \ldots, s_k$), ideally including the correct answer (s_i). To better approximate a realistic crossword-solving scenario and to further constrain the search space, each clue (c_i) is accompanied by the character length of its target answer (s_i). For example, given the clue and target length *"Un passo indietro nel tempo", 4*[1], the system should output a list of candidate answers that ultimately includes the correct solution, *"ieri"*[2].

Subtask 2 focuses on the autonomous solving of Italian crossword grids. Participants are provided with a set of empty crossword grids ($G = G_1, G_2, \ldots, G_k$), where each grid ($G_i$) is associated with a list of clues. Each clue is annotated with the $((x, y))$ coordinates of the starting cell of its corresponding answer in the grid, as well as its direction—either across (*orizzontale*) or down (*verticale*). Each crossword grid (G_i) is represented as an ($R \times R$) matrix, in which each cell is either empty or blocked by a black square. The developed systems are required to autonomously fill the grid with suitable answers,

EVALITA 2026: 9th Evaluation Campaign of Natural Language Processing and Speech Tools for Italian, Feb 26 – 27, Bari, IT
pierpaolo.basile@uniba.it (P. Basile)
https://www.uniba.it/it/docenti/pierpaolo-basile (P. Basile)
0000-0002-0545-1105 (P. Basile)

[1]In English: "A step back in time"
[2]In English: "Yesterday"

producing a fully or partially completed crossword that enforces character-level consistency at word intersections and maximizes the number of correctly placed solutions.

A language game has already been proposed in two editions of EVALITA (2018 and 2020) [3, 4]. The game was based on the TV game "La Ghigliottina", which involves a single player, who is given a set of five words - the clues - each linked in some way to a specific word that represents the unique solution of the game. Words are unrelated to each other, but each of them has a hidden association with the solution. Once the clues are given, the player has one minute to find the solution. The presence of other language-based tasks in previous editions of EVALITA supports the interest and vitality of the research community in this kind of task.

The Cruciverb-IT task is particularly timely, as language games have recently gained attention as effective benchmarks for probing the reasoning and interpretative capabilities of Language Models (LMs). Among them, crossword puzzles constitute a particularly demanding task, as they require not only lexical and syntactic competence but also cultural knowledge, lateral thinking, and the ability to resolve ambiguity and polysemy [5, 6, 7, 8]. Crossword solving involves complex semantic and pragmatic inference, making it a valuable setting for evaluating deeper language understanding beyond surface-level similarity matching.

Before the emergence of modern LMs, crossword-solving systems primarily relied on retrieval-based approaches and shallow lexical or semantic features [9, 10]. Subsequent work introduced more structured linguistic information, such as lexical resources, syntactic cues, and ranking strategies, to improve clue–answer matching in Italian [11, 12]. However, these approaches remain limited when addressing clues that require non-literal interpretation, wordplay, or semantic ambiguity resolution.

Despite significant progress by Large Language Models (LLMs), solving crossword puzzles remains challenging, especially for under-resourced languages such as Italian [13]. Our primary motivation is to build an effective solution without relying on closed LLMs or fine-tuning open LLMs with larger numbers of parameters. For that reason, we choose to work with an encoder-decoder architecture specific to Italian (IT5 [14]) with a small number of parameters compared to current small-to-medium LLMs. Moreover, to fill the crossword, we want to use a dynamic algorithm that updates the list of candidate solutions based on the characters already inserted in the grid. For that purpose, we train IT5, taking partial answers into account when we already know some characters. All the methodological details are reported in Section 2, while results and discussion are provided in Section 3.

2. Description of the System

Our approach is based on the idea of solving a crossword as humans do. Humans usually approach a crossword puzzle by first scanning the clues and filling in the answers they immediately recognize. These early answers provide anchor letters that help narrow down the possibilities for more difficult clues. Solvers often alternate between across and down clues, using intersecting letters to test hypotheses and eliminate incorrect options. Over time, the grid becomes increasingly constrained, allowing previously ambiguous clues to be solved through cross-checking and incremental refinement.

Generally, several approaches in the literature are based on the idea of providing a list of candidate solutions and then using it to fill the crossword using constraint programming or similar techniques. Our idea is to generate solutions based on the clue and the partial solution, if some characters are already available in the crossword. For these reasons, we train an encoder-decoder model that takes as input not only the clue but also the partial solution. For example, given the clue *"Un passo indietro nel tempo"*[3] and the solution *"ieri"*[4], we generate all the examples that involve all the possible partial solutions obtained by masking one or more characters.

The Table 2 reports a portion of all the possible partial solutions of the word "ieri". Given a word composed of n characters, all the possible partial solutions are 2^n. We also add the number of missing characters in the solution and the solution length to the encoder's input text.

[3] In English: "A step back in time"
[4] In English: "Yesterday"

Partial solution	Missing chars
ieri	0
i_ri	1
i__i	2
_e__	3

Table 1
Example of partial solutions of the word: “ieri”

Using this approach, we can prompt the model even when we know some of the final solution’s characters. Obviously, we also prompt the model where no characters are known. By following this idea, we built our approach. The whole pipeline is composed of several steps:

1. produce training data using the initial training set provided by the task organizers;
2. train the encoder-decoder model on training data;
3. define a strategy for selecting the best solution given the current crossword and the list of clues;
4. define a solution strategy for filling the crossword.

For step 3, we rely on the score produced by the encoder-decoder strategy during generation, while for step 4, we try different solution search algorithms, such as greedy, A*, and beam search. After a preliminary evaluation on the development set provided by the organizers, we chose beam search because it offers the best compromise between performance and execution time. In this report, we report only the details about the beam search implementation. The following subsections provide more details on each step.

2.1. Encoder-Decoder Training

Our idea is to generate for each pair of clue and answer several training examples in which some characters of the original answer are masked using the underscore character. Formally, given an answer of n characters, all the possible partial solutions that we can obtain by masking one or more characters are equal to 2^n. Following this approach, starting from the original training data of 374,766 pairs, we obtain 16,254,904 training examples. Each example is composed of: 1) the partial solution obtained by masking some characters (*partial_answer*); 2) the number of missing characters in the answer (*missing_chars*); 3) the correct answer (*answer*) and 4) the total number of characters that compose the correct answer (*answer_length*). All training data are stored in a JSON line file, as shown in Listing 1.

Listing 1: "Example of training data."

```
{"partial_answer": "_di", "missing_chars": 1, "answer": "idi", "answer_length": 3
    , "clue": "Giorni di met\u00e0 mese nell'antica Roma"}
{"partial_answer": "_l__ari", "missing_chars": 3, "answer": "alamari", "
    answer_length": 7, "clue": "Possono sostituire le mostrine"}
...
```

We train three models. The first two models are trained using the dataset composed of partial answers. The difference between the two models lies in how the input text is represented:

- **Model 1:** the input is formatted using special tokens that are added to the tokenizer in the following way:

  ```
  [CIT_MISS_LENGTH] {0} [CIT_LENGTH] {1} [CIT_CLUE] {2} [CIT_ANSWER] {3}
  ```

  where {0} is the number of missing characters, {1} is the answer length, {2} is the clue and {3} is the partial answer. The added special tokens are: `[CIT_MISS_LENGTH]`, `[CIT_LENGTH]`, `[CIT_CLUE]` and `[CIT_ANSWER]`.
- **Model 2:** the input is formatted using plain text without the usage of special tokens in the following way:

```
"Trova la soluzione dove _ indica un carattere mancante. Caratteri mancanti
    : {0}. Lunghezza soluzione: {1}. Soluzione parziale {2}. Indizio: {3}"
```

where {0} is the number of missing characters, {1} is the answer length, {2} is the partial answer and {3} is the clue.

Finally, **Model 3** is trained only on the original dataset without using a partial solution. The input text is formatted as for the Model 2:

```
"Trova la soluzione. Lunghezza soluzione: {0}. Indizio: {1}"
```

where {0} is the answer length and {1} is the clue. For this kind of input, where partial answers are not exploited, we trained the model without special tokens, as we observed on the organizers' validation data that worse results were obtained when they were used. All the results on validation and test data are reported in Section 3.

For all three models, the decoder's output corresponds to the correct answer. All models are based on IT5 Large[5] [15], an encoder-decoder transformer model pretrained specifically for Italian. IT5 is trained on a cleaned web-crawled Italian corpus including more than 40 billion words. We did not add further training data, such as dictionary definitions. Our idea is to build the model solely from the provided data.

For all the models, we use the following parameters: learning rate = 5e-5, batch size = 32, weight decay = 0.01. We use different numbers of epochs: the models that exploit the complete training data with partial answers are trained on a single epoch, while model 3 is trained on 10 epochs. We use a single GPU NVIDIA RTX A6000 with 48GB of VRAM. On the validation set, Model 1 performs worse than Model 2; therefore, we exclude it.

2.2. Fill the Grid

The strategy used to fill the grid relies on previous models to produce, for each clue, a list of candidates based on the clue and the characters already present in the crossword. The list of candidates is not computed only once when the crossword is empty, but is recalculated at each step, using characters added to the crossword during the filling process. To exploit the solution space, we implement a beam search algorithm that selects the best k alternatives at each step and follows them in the next step. One main problem is choosing the best candidate answer at each step to put into the crossword. The idea is to follow the human approach: select the answer with the highest confidence. As a first approach, we rely on the score generated by the beam search of the IT5 decoder during the answer generation. We observe that this approach does not yield good results; therefore, we chose to exploit the correct grids (solved crosswords) provided by the organizers as training data.

The idea is to build a binary classifier that takes a candidate solution as input and predicts whether it is a correct answer. Several features represent the candidate solution:

1. the number of rows and columns in the grid;
2. the grid completeness that is the ratio between filled cells and the total number of cells to fill in the grid;
3. the starting row and column of the current clue where the candidate solution belongs;
4. the number of missing characters, this feature is used to track if some characters of the solution are already known;
5. the ratio between missing characters and the answer length;
6. the direction: across or down;
7. 1 if the candidate solution belongs to the list of answers observed in training data, 0 otherwise;
8. the beam search score provided by the decoder during the generation.

[5] https://huggingface.co/gsarti/it5-large

For each grid in the training data, we simulate a game in which clues are selected at random until the grid is completed. At each step, when a clue is selected, we generate training data for each candidate solution to the current clue. For IT5, we use a beam size of 100 at each step to produce a list of almost 100 candidates for each clue. The generated list may be less than 100 since we remove all candidates that contain spaces, violate the grid constraints or whose lengths differ from the answer length. This approach is the same as the one used during the crossword filling in the testing phase, since the answer length is a known parameter when solving the crossword. The training dataset contains 500 grids, and the number of generated examples used to train the binary classifier is 65,863. Obviously, the majority of the examples are classified as negative, since the incorrect candidates outnumber the correct ones. We use XGBoost as the model behind our classifier and, to account for the dataset's imbalance, we set the `scale_pos_weight` parameter to the ratio of negative to positive classes. The other XGBoost parameters are set as follows: `n_estimators=5000`, `learning_rate=0.05` and `max_depth=6`.

To test the binary classifier's performance, we compute its accuracy on the validation grids provided by the organizers. The validation contains 50 grids and generates 14,502 evaluation examples. The performances are summarized in Table 2.2. The classifier obtains satisfactory results. Nevertheless, it struggles with the positive class, which is expected given the dataset's high imbalance. The number of false positives is 765 (5.2%), while the number of false negatives is 215 (1.5%).

class	precision	recall	f1-score	support
0	.98	.94	.96	13,084
1	.61	.85	.71	1,418
accuracy	.93	14,502		
macro avg	.80	.89	.84	14,502
weighted avg	.95	.93	.94	14,502

Table 2
Performance of the binary classifier used to classify candidate answers.

These performances can be considered acceptable, since the beam search algorithm that exploits the candidate list does not select a single answer from the list, but instead ranks candidates according to the probability of each class provided by the classifier. In particular, if the classifier predicts class 1, the score is equal to $1 + P(class_1)$; otherwise, the score is $1 - P(class_0)$.

When we try to solve the crossword, at each step we compute and rank the list of candidate answers for all unfilled words in the grid. Then, we select the unfilled word with the candidate answer that obtains the highest score and generate a beam for each candidate answer of that word to exploit more solutions. These beams are added to the list of possible beams, and only the k most promising are retained. The sum of the current score and scores of all selected answers from the previous steps ranks the beams.

When we produce the list of candidate answers for each unfilled word, we take into account several events:

- we remove all candidates that contain spaces;
- we remove all candidates that do not respect the expected length;
- we remove all candidates that do not respect the grid constraints, for example, if crossing cells are already filled with a different character.

During the candidates generation for a specific unfilled word, the generated list may be empty; in this case, a fake solution composed of spaces[6] with a very low score ($1e-6$) is added to the list. The spaces are not considered when the algorithm checks whether a candidate satisfies the grid constraints; this gives the algorithm greater flexibility, allowing it to proceed even when no candidate solutions are available. This means that it is possible to generate incomplete crosswords.

We produce two variants of beam search: the former exploits only candidate solutions labeled as positive by the binary classifier, whereas the latter retains all candidates.

[6]We retain crossing characters already present in the grid.

It is essential to underline that at each step of the beam search, if characters are already available in the grids, they are used to form the partial solution, which is then fed to the encoder-decoder to generate a list of possible candidates.

In all experiments, we set the encoder-decoder beam search to 100, while the grid-filling algorithm uses a beam search of 20. We select these parameters according to a preliminary evaluation on the validation set. The only parameter that we change concerns the use of candidate answers labeled as negative. In submitted runs, we exclude the model trained with special tokens (model 1) because it performs the worst.

3. Results and Discussion

In this section, we report the results obtained in each subtask.

For **subtask 1**, we perform two runs. The first run (RUN1) relies on the encoder-decoder trained on the original dataset, while the second run (RUN2) uses the model trained on partial answers. For both models, we use a beam size of 100 and retain only the first 10 answers that do not contain spaces and that are within the expected answer length, since the subtask allows a maximum of 10 solutions per clue. It is essential to note that when generating the model input in RUN2, we mask all characters in the partial solution and set the number of missing characters to the answer length.

Metrics for subtask 1 are $acc@1$ and $acc@10$, which are the accuracy in retrieving the correct solution word given the corresponding clue, considering the top 1 and 10 candidates produced by the system, and Mean Reciprocal Rank (MRR), that is, the average of the reciprocal ranks of the first relevant item across all clues. Table 3 shows the results on subtask 1. We observe that the model trained on partial answers achieves a remarkable gain over the model trained only on the complete answer. However, the best performance (RUN2) is close to the organizers' baseline and far from the best system. We have no details about the best system's approach, but we want to underline that we do not exploit external data, such as dictionaries or large language models, whose use is allowed.

System	acc@1	acc@10	MRR
Best system	0.69	0.83	0.72
UNIBA RUN2	0.43	0.59	0.47
Baseline	0.4	0.62	0.46
UNIBA RUN1	0.36	0.54	0.41

Table 3
Results of the subtask 1.

Regarding **subtask 2**, we provide two runs. RUN1 takes into account all the candidates, while RUN2 considers only the candidates labeled as positive by the binary classifier. Metrics for subtask 2 are: $char\_acc$, which is the accuracy in inserting the correct characters in the correct slots; $word\_acc$, the accuracy in inserting the correct word in the correct slots; $full\_match\_accuracy$, the accuracy in solving the entire grid. A partially filled grid will be evaluated, counting empty squares as errors. Results are reported in Table 3. Essentially, both runs provide similar results. It is probably because candidate answers labeled as negative receive low scores and are discarded during beam search. This happens where candidates labeled as positive exist for each unfilled word. In fact, negative answers are

System	char_acc	word_acc	full_match_accuracy
Best system	0.92	0.85	0.34
UNIBA RUN2	0.82	0.67	0.16
UNIBA RUN1	0.82	0.66	0.16
Baseline	0.73	0.58	0.08

Table 4
Results of the subtask 2.

a fallback when no positive candidates are available. In testing data, this case is probably rare, and the two runs provide similar results.

Looking at the results, we observe a clear gain over the baseline. Compared to the baseline, we achieve a remarkable improvement in performance, even though our system is very close to it on subtask 1. The baseline leverages the output of subtask 1, combined with an additional module that optimizes for a solution by maximising satisfied constraints while respecting the grid's hard constraints. By treating crossword puzzles as a weighted Max-SMT problem, as partially described in [16]. The final problem formulation is passed to the Z3 optimizer, which satisfies all hard constraints and maximizes the weighted satisfaction of soft constraints. This means that our approach, which exploits the characters already present in the grids to build the candidate list of answers at each step, can improve overall performance, even if the accuracy when all characters are unknown is similar to the baseline.

The gap relative to the best system is also reduced, demonstrating the effectiveness of our strategy for filling the grid using the available answers. Moreover, beam search can deliver remarkable results even though its complexity is lower than the baseline, and the entire solution space is not explored, which does not guarantee finding the best solution.

It is also interesting to analyze the system's accuracy in solving crosswords as a function of the grid's size. Table 3 shows the accuracy of each system in solving grids of different sizes. We observe that no system can solve grids larger than 9, and UNIBA's performance on a grid of size 9 is equal to that of the best system. However, the best system delivers very good results on small grids. The baseline provides a correct grid only for size 5.

System	5x5	7x7	9x9	11x11	13x13
Best System	0.9	0.6	0.1	0	0
UNIBA RUN1	0.4	0.3	0.1	0	0
UNIBA RUN2	0.4	0.3	0.1	0	0
Baseline	0.4	0	0	0	0

Table 5
Accuracy at different grid sizes.

Finally, our approach delivers remarkable results that support our motivations behind the designed methodology. The approach seems to be penalized by the encoder-decoder model's performance when all the solution's characters are unknown. This can depend essentially on two factors: the quality of the base model (IT5) and the use of only the training data provided by the organizers, which limits the model's ability to generalize. It could be interesting to test the approach on other languages, for example, English, using the T5 family of models, or to test multilingual T5 models on Italian instead of relying on IT5. The quality of the training data can be improved by adding definitions from dictionaries, as already done in other approaches to solving Italian crosswords.

3.1. Additional Experiments

After the task submission deadline and after the test becomes available, we perform additional experiments. In particular, we aim to improve the beam search algorithm for grid filling. One limitation is that the beam search at each step selects the unfilled word with the best candidate answer, then expands the search by exploring the other candidates for that word. This approach limits the exploration of other unfilled words. For that reason, we modify the beam search and compute the following grids at each step, taking into account all candidates for all unfilled words. Then we retain only the best k beams. In this way, we try to explore more unfilled words at each step rather than focusing only on the unfilled word with the best candidate. However, this approach has a drawback: using a fixed beam width can limit the exploration of unfilled words in large grids. We overcame this limit by using a dynamic beam width. We use a fixed factor bw_f and multiply it by the number of unfilled words in the grid. Using this approach, it is possible to explore more grids at the start when more words are unfilled. Using a $bw_f = 5$, we obtain an increment of solved grids from 0.16 to 0.2 while $char\_acc$ and $word\_acc$ remain

the same (0.81 and 0.67). In particular, the number of solved 5x5 grids increases from 4 to 6, achieving an accuracy of 0.6, while the accuracy for other grid sizes remains the same. This result can be explained by the encoder-decoder's low performance, which is a bottleneck in the search for the best solution.

Finally, we conduct an experiment to understand how performance changes as the number of missing characters in the correct answer increases. Using the validation data from subtask 1, we build an evaluation dataset that considers all possible partial solutions derived from the correct answer by removing characters. Since this dataset is very large and a complete evaluation would take a long time, we consider a random 25% subset. Results in Table 6 show that performance increases with the percentage of known characters relative to the length of the correct answer. These results prove the effectiveness of our model in finding the correct answer when some characters are already present in the crossword.

% known chars.	acc@1	acc@10	MRR
<10	0,0579	0,2828	0,1017
<20	0,0698	0,3231	0,1115
<30	0,0759	0,3488	0,1222
<40	0,0877	0,396	0,1372
<50	0,0892	0,4048	0,1418
<60	0,0989	0,4392	0,1512
<70	0,1007	0,4439	0,1525
<80	0,1159	0,5215	0,1851
<90	0,1385	0,561	0,1849

Table 6
Performance according to the percentage of known characters of the correct answers.

4. Conclusions and Future Works

In this paper, we present the UNIBA system submitted to the Cruciverb-IT task at EVALITA 2026, addressing both clue answering and crossword grid completion in Italian. Our approach combines an encoder–decoder model pretrained for Italian with a dynamic, search-based strategy that incrementally exploits partial solutions during grid filling. By explicitly modeling partially known answers and updating candidate generation as the grid evolves, the proposed system closely mimics human behavior in crossword solving.

Experimental results show that training the encoder–decoder model on partially masked answers leads to consistent improvements in clue answering performance compared to a model trained only on complete answers. More importantly, the proposed grid-filling strategy significantly outperforms the official baseline on all metrics for subtask 2, demonstrating that dynamically exploiting intersecting letters is crucial for effective crossword completion. These results are achieved without relying on external lexical resources, dictionaries, or large proprietary language models, highlighting the effectiveness of lightweight, task-specific architectures combined with structured search strategies.

At the same time, our analysis reveals several limitations. The overall performance remains strongly influenced by the quality of the candidate answers generated when no characters are initially available, which represents a bottleneck for both subtasks. Moreover, despite using beam search, solving larger grids (11×11 and above) remains challenging, suggesting that more expressive candidate generation or stronger global reasoning is required to fully explore the solution space.

Future work will focus on improving the encoder–decoder component, for instance, by augmenting the training data with high-quality external resources such as dictionary definitions or encyclopedic descriptions, which are known to be beneficial for crossword solving. Another promising direction is evaluating multilingual or larger T5-based models to assess whether improved semantic representations can mitigate the limitations observed when partial information is unavailable. From a search perspective, further refinements of the grid-filling algorithm—such as adaptive beam widths, more informed clue-selection strategies, or tighter integration between generation and constraint satisfaction—could improve

scalability to larger grids. Finally, extending the proposed framework to other languages and crossword datasets would help assess its generality and robustness beyond the Italian setting.

Acknowledgments

We acknowledge the support of the PNRR project FAIR - Future AI Research (PE00000013), Spoke 6 - Symbiotic AI (CUP H97G22000210007) under the NRRP MUR program funded by the NextGenerationEU.

Declaration on Generative AI

During the preparation of this work, the author(s) used Grammarly to: Grammar and spelling check, Paraphrase and reword, and Improve writing style.

References

[1] C. Ciaccio, G. Sarti, A. Miaschi, F. Dell'Orletta, M. Nissim, Cruciverb-it @ evalita 2026: Overview of the crossword solving in italian task, in: Proceedings of the Ninth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2026), CEUR.org, Bari, Italy, 2026.

[2] F. Cutugno, A. Miaschi, A. P. Aprosio, G. Rambelli, L. Siciliani, M. A. Stranisci, Evalita 2026: Overview of the 9th evaluation campaign of natural language processing and speech tools for italian, in: Proceedings of the Ninth Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop (EVALITA 2026), CEUR.org, Bari, Italy, 2026.

[3] P. Basile, M. De Gemmis, L. Siciliani, G. Semeraro, Overview of the evalita 2018 solving language games (nlp4fun) task, EVALITA Evaluation of NLP and Speech Tools for Italian 12 (2018) 75.

[4] P. Basile, M. Lovetere, J. Monti, A. Pascucci, F. Sangati, L. Siciliani, et al., Ghigliottin-ai@ evalita2020: Evaluating artificial players for the language game "la ghigliottina", in: CEUR WORKSHOP PROCEEDINGS, AILC-Associazione Italiana di Linguistica Computazionale, 2020, pp. 345–348.

[5] E. Wallace, N. Tomlin, A. Xu, K. Yang, E. Pathak, M. Ginsberg, D. Klein, Automated crossword solving, in: S. Muresan, P. Nakov, A. Villavicencio (Eds.), Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Dublin, Ireland, 2022, pp. 3073–3085. URL: https://aclanthology.org/2022.acl-long.219/. doi:10.18653/v1/2022.acl-long.219.

[6] J. Rozner, C. Potts, K. Mahowald, Decrypting cryptic crosswords: Semantically complex wordplay puzzles as a target for nlp, Advances in Neural Information Processing Systems 34 (2021) 11409–11421.

[7] S. Saha, S. Chakraborty, S. Saha, U. Garain, Language models are crossword solvers, in: Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), 2025, pp. 2074–2090.

[8] A. Sadallah, D. Kotova, E. Kochmar, What makes cryptic crosswords challenging for llms?, in: Proceedings of the 31st International Conference on Computational Linguistics, 2025, pp. 5102–5114.

[9] M. Ernandes, G. Angelini, M. Gori, et al., Webcrow: A web-based system for crossword solving, in: AAAI, 2005, pp. 1412–1417.

[10] G. Angelini, M. Ernandes, M. Gori, Solving italian crosswords using the web, in: Congress of the Italian Association for Artificial Intelligence, Springer, 2005, pp. 393–405.

[11] G. Barlacchi, M. Nicosia, A. Moschitti, A retrieval model for automatic resolutionof crossword puzzles in italian language, in: Proceedings of the First Italian Conference on Computational Linguistics CLiC-it 2014 & and of the Fourth International Workshop EVALITA 2014: 9-11 December 2014, Pisa, Pisa University Press, 2014, pp. 33–37.

[12] A. Moschitti, M. Nicosia, G. Barlacchi, Sacry: syntax-based automatic crossword puzzle resolution system, in: Proceedings of ACL-IJCNLP 2015 System Demonstrations, 2015, pp. 79–84.
[13] C. Ciaccio, G. Sarti, A. Miaschi, F. Dell'Orletta, Crossword space: Latent manifold learning for italian crosswords and beyond, in: Proceedings of the Eleventh Italian Conference on Computational Linguistics (CLiC-it 2025), 2025, pp. 245–255.
[14] G. Sarti, M. Nissim, It5: Text-to-text pretraining for italian language understanding and generation, in: Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024), 2024, pp. 9422–9433.
[15] G. Sarti, M. Nissim, IT5: Text-to-text pretraining for Italian language understanding and generation, in: N. Calzolari, M.-Y. Kan, V. Hoste, A. Lenci, S. Sakti, N. Xue (Eds.), Proceedings of the 2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING 2024), ELRA and ICCL, Torino, Italy, 2024, pp. 9422–9433. URL: https://aclanthology.org/2024.lrec-main.823.
[16] S. Kulshreshtha, O. Kovaleva, N. Shivagunde, A. Rumshisky, Down and across: Introducing crossword-solving as a new NLP benchmark, in: S. Muresan, P. Nakov, A. Villavicencio (Eds.), Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), Association for Computational Linguistics, Dublin, Ireland, 2022, pp. 2648–2659. URL: https://aclanthology.org/2022.acl-long.189/. doi:10.18653/v1/2022.acl-long.189.

A. Online Resources

The following resources are available:

- IT5 models for retrieving answers where partial solution is available
- IT5 models for retrieving answers
- Code