Guaranteed Generation from Large Language Models

Minbeom KIm, Thibaut Thonet, Jos Rozen, Hwaran Lee, Kyomin Jung, Marc Dymetman

This work is done during Minbeom's internship at NAVER Labs Europe

Goals of Guaranteed Text Generation

LLMs will be increasingly adopted in critical decision-making.

The more important the decisions an LLM has to make, even a slight leakage of undesired attributes can have severe consequences.

We want 💯% strict guarantees on our requirements!!!

Three Goals of Guaranteed Generation

1. How can we **guarantee that all generated sequences** from LLMs meet specific constraints or requirements?
2. How can we achieve this while **preserving the original model's useful distribution as much as possible**?
3. How can we simultaneously obtain the two previous properties at run-time with a **limited inference cost**?

Notations

- Base LLM: $a(y)$
  - Sampling: $y \sim a(y)$
- The guarantee: any type of hard constraint $b(y)$
  - $b(y) \in \{0, 1\}$
  - $b(y) = 1$: guarantee satisfaction
  - $b(y) = 0$: guarantee violation
- The guarantee can be a simple or complex predicate
  - E.g. $b(y)$ is a toxicity detector (0 means toxic, 1 means non-toxic)
  - E.g. $b(y)$ is a verifier of certain keywords (0 means 'does not contain keyword', 1 means 'contains keyword')
  - E.g. $b(y)$ also can be complex checker: Personal information leakage to unauthorized people
- Guaranteed sampler: sampler that **never** violates the $b(y)$ guarantee.

Ideal distribution g for Guaranteed Generation

- We want a guaranteed sampler that **distorts** $a(y)$ **as little as possible** while satisfying $b(y)$, we define the target model as

$$g(y) \propto a(y)b(y) \qquad \text{# filtered model}$$

$$g(y) = \frac{1}{Z}a(y)b(y), \qquad \text{# target model}$$

$$Z = \sum_{y} a(y)b(y) \qquad \text{# normalizer}$$

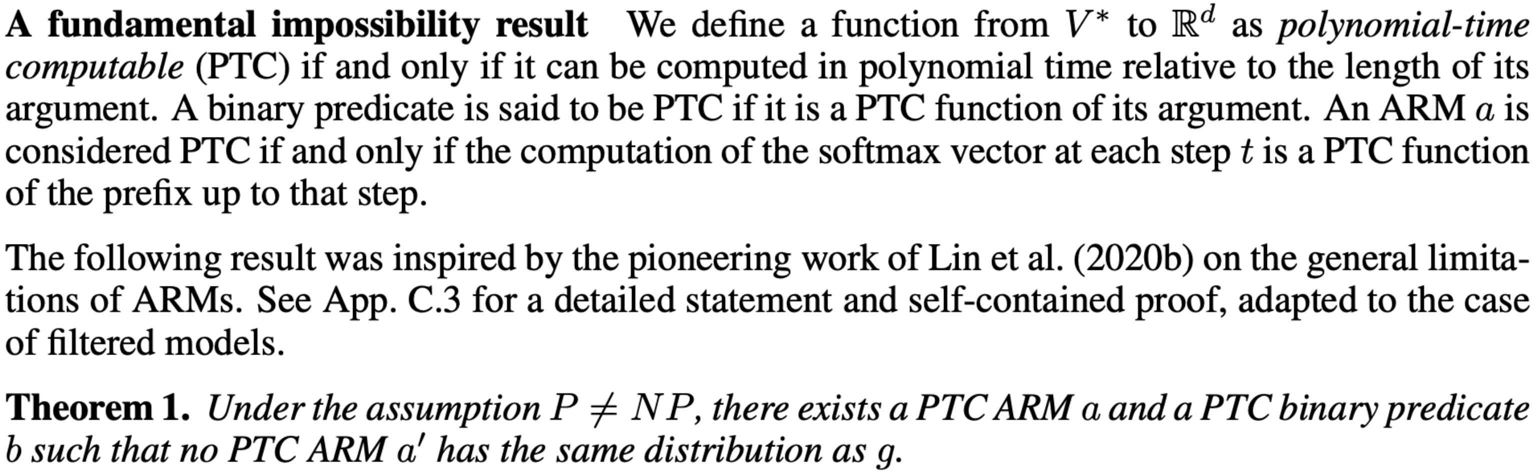
- $g(y) = a(y \mid b(y) = 1)$
- Our target model g **minimally** deviates from a to satisfy the constraint b, which can be measured by $KL(g \mid a)$.
- That is why we target $g(y)$!!!

Impossibility to obtain g with training only

- We want to obtain **exact sampler:** $y \sim g(y)$. Can we obtain it with autoregressive models?

- First, it is impossible in general to find autoregressive model $a'(y) = g(y)$, $(P \neq NP)$

- Second, fine-tuning cannot produce an exact sampler.



A fundamental impossibility result We define a function from V^* to $\mathbb{R}^d$ as *polynomial-time computable* (PTC) if and only if it can be computed in polynomial time relative to the length of its argument. A binary predicate is said to be PTC if it is a PTC function of its argument. An ARM a is considered PTC if and only if the computation of the softmax vector at each step t is a PTC function of the prefix up to that step.

The following result was inspired by the pioneering work of Lin et al. (2020b) on the general limitations of ARMs. See App. C.3 for a detailed statement and self-contained proof, adapted to the case of filtered models.

Theorem 1. *Under the assumption $P \neq NP$, there exists a PTC ARM a and a PTC binary predicate b such that no PTC ARM a' has the same distribution as g.*

Rejection sampling to obtain g

To obtain Ideal distribution $g(y) = \begin{cases} 0, & b(y) = 0 \\ \frac{1}{Z}a(y), & b(y) = 1, \end{cases}$

- Rejection sampling is an exact sampler!

1) $y \sim a(y)$
2) If $b(y) = 1$, return y; else, go to 1

- Acceptance rate (AR_a) of this system is $AR_a = \sum_y a(y)b(y) = Z$
- If AR_a is high enough to directly use, then perfect!
- Unfortunately, AR_a is low for various general use cases = heavy inference costs.
- We need a more efficient approach!

What are better options?

- We target **guaranteed** samplers under $b(y)$ and also **almost exact** sampler to $g(y)$
- Option 1: Construct faster system with autoregressive model $a'(y)$, but give up on the guarantees
- Option 2: Approximate $g(y)$ by MCMC sampling techniques, but it also requires heavy inference costs.
- Option 3: Rejection sampling $a'(y)$ on proxy target $g'(y) \propto a'(y)b(y)$

$$1)\ y \sim a'(y)$$

$2)$ If $b(y) = 1$, return y;   else, go to 1

- Yes, it is **guaranteed** and **exact** sampler of $g'(y)$, but not of $g(y)$.
- So, critical problem is the '**quality**' of $g'(y)$ toward $g(y)$.
- If $KL(g\,|\,g')$ is too high, we could try to improve $a'(y)$ so that this divergence becomes smaller.
- **New Goals with proposal** a': High quality of $KL(g\,|\,g')$ with good efficiency $AR_{a'}$

Distributional relationship behind approximated sampler

- GUARD's key theorem for optimizing proposal distribution $a'(y)$

Theorem 2: $\underbrace{KL(g\,|\,a')}_{objective} = \underbrace{-\log AR'_a}_{efficiency} + \underbrace{KL(g\,|\,g')}_{closeness},$

- We prove that $KL(g\,|\,a')$ **minimization** jointly optimize better **efficiency** and distributional **closeness** to our sampling target.

- Conclusion: Let's **minimize** $KL(g\,|\,a')$ for **guaranteed** and **almost exact sampler** to target distribution g!!!

Baselines for Guaranteed Generation

1. Constraint-aware Prompting (CAP): Give prompts to encourage the output y to respect the given constraint $b(y) = 1$

+: Do not need any sample from g.

- : There is no guarantee to minimize $KL(g \,|\, a') = KL(g \,|\, a, CAP)$

2. Supervised Fine-Tuning (SFT)

   1) Sample a large number of $y \sim a$, then filter them only $b(y) = 1$ to create dataset representing g.
   2) Then, fine-tune a with this dataset to train proposal a'_θ.

+: Theoretically guaranteed to minimize $KL(g \,|\, a'_\theta)$

- : We need to sample a lot of $y \sim a$ to create a sufficient dataset.

Baselines for Guaranteed Generation

3. Distributional Policy Gradient (DPG): with proposal a'_θ,

 1) Sample $y \sim a'_\theta$ with adaptive proposal, then filter them with $b(y) = 1$.
 2) Then, train a'_θ with DPG loss $\nabla_\theta KL(g \,|\, a'_\theta) = -\mathbb{E}_{y \sim g} \nabla_\theta \log \pi_\theta(y) = -\mathbb{E}_{y \sim \pi_\theta} \dfrac{g(y)}{\pi_\theta(y)} \nabla_\theta \log \pi_\theta(y)$

+: Theoretically guaranteed to minimize $KL(g \,|\, a'_\theta)$
+: As the proposal a'_θ approaches g, the frequency and quality of gradient updates increase, making it more efficient.

- : Still inefficient in the early stage when $a'_\theta \simeq a$.

4. Warm-start DPG (adopted as GUARD Training algorithm)

 1) Sample $y \sim a(\cdot \,|\, CAP)$, then filter them with $b(y) = 1$ and fine-tune a with this dataset to make initial point a'_θ.
 2) Start DPG training on warm-starting point.

+: Theoretically guaranteed to minimize $KL(g \,|\, a'_\theta)$
+: As the proposal a'_θ approaches g, the frequency and quality of gradient updates increase, making it more efficient.
+: Skip the slow early exploration stage where training samples are scarce.

Experimental Setting

- Unconditional text generation under lexical constraints

Model a: Gemma-2B

Goals: Starting with the ⟨bos⟩ token, a generates a text y of 30 tokens with keywords “amazing”

Binary constraint b: $b(y) = 1$ iff y contains the string “amazing”

AR_a of b ~ 0.0023

- Conditional text generation under sentiment reversal

Model a: GPT-2 finetuned on ROCStories Sentiment scorer b_{score}: 🤗 michellejieli/emotion_text_classifier

Goals: Start with very negative story opening X ($b_{score}(X) < 0.05$), complete positive ending story $y = [Y; Z]$.

Binary constraint b: $b(y) = 1$ iff the final sentence Z is very positive $b_{score}(Z) > 0.98$.

AR_a of b ~ 0.005

Experimental Results

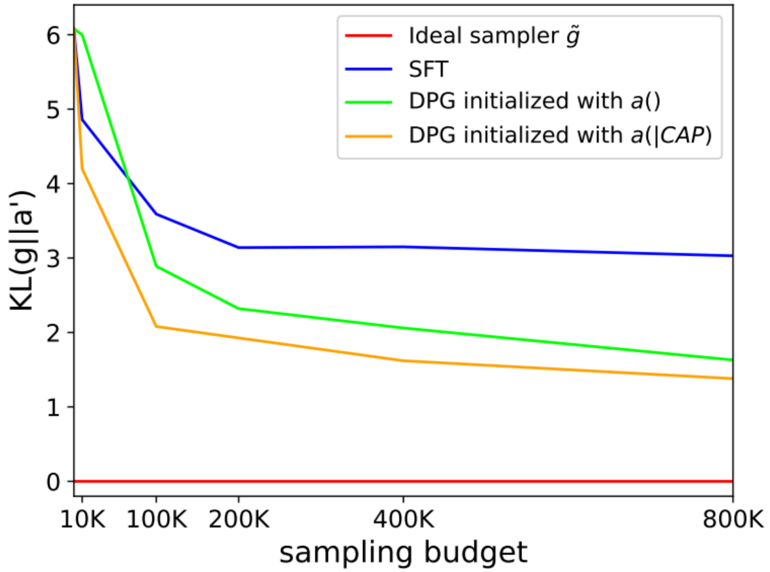


Figure 3: Evolution of $\mathrm{KL}(g||a')$ as a function of the number of samples used for training with lexical constraints.

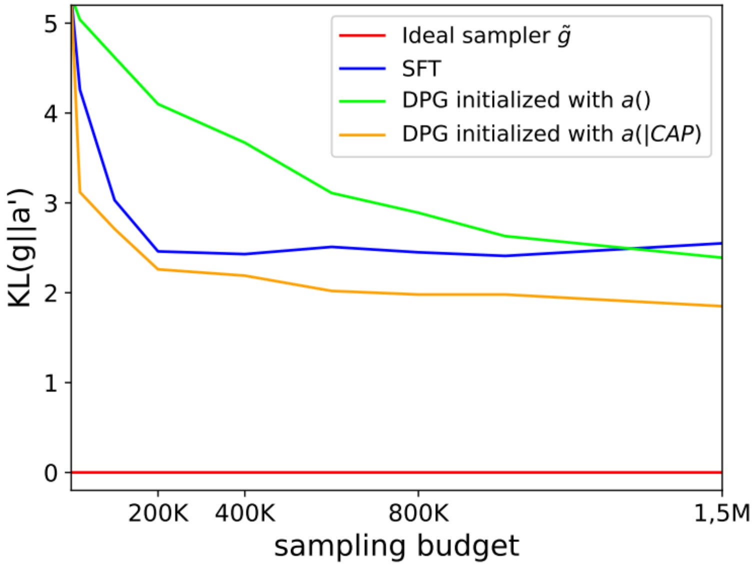
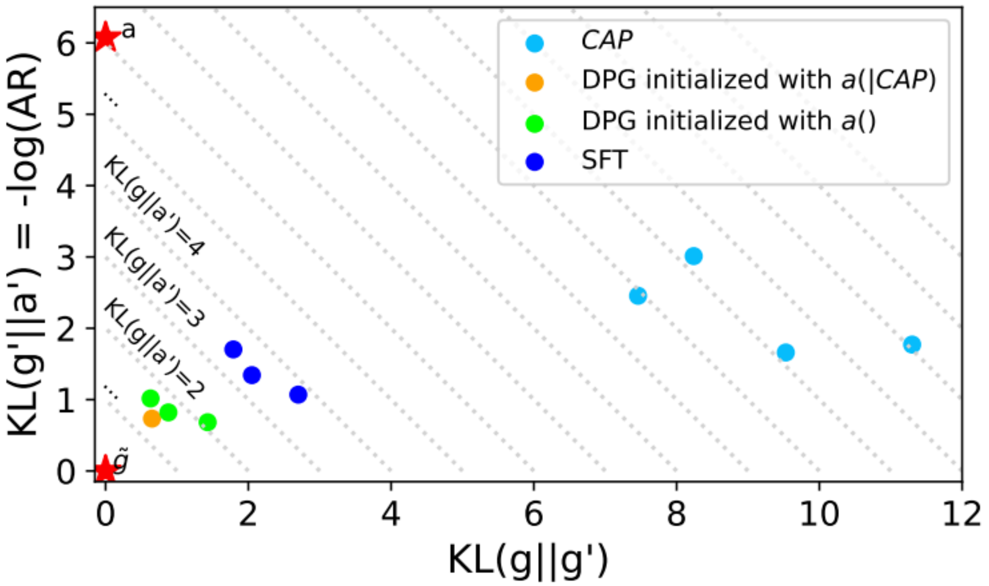
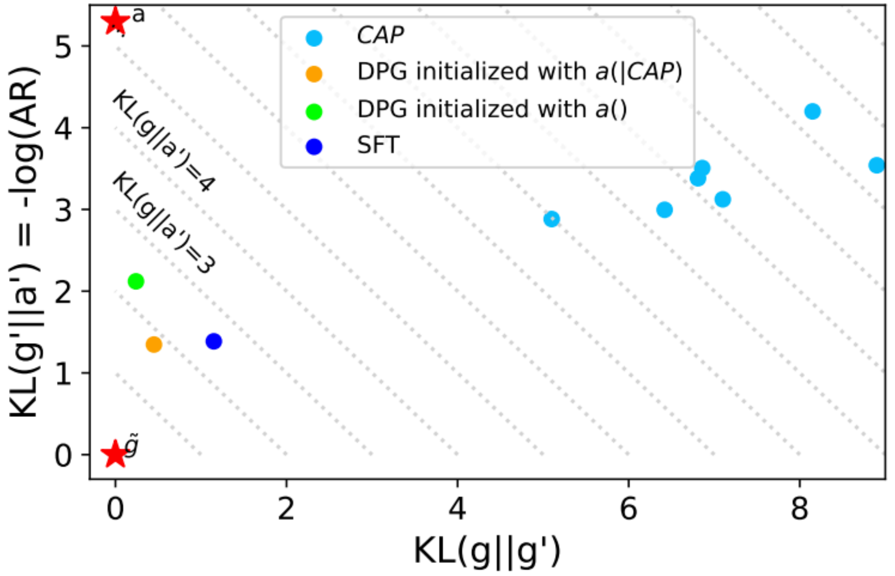


Figure 5: Evolution of $\mathrm{KL}(g||a')$ as a function of the number of samples used for training in sentiment reversal.

- With adaptive proposal a', DPG shows better optimization than SFT.
- Warm-start DPG avoid inefficient early stage with off-policy sampler $a(\cdot \mid CAP)$.

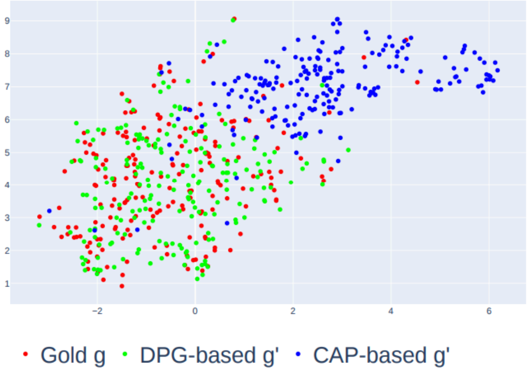
Experimental Results

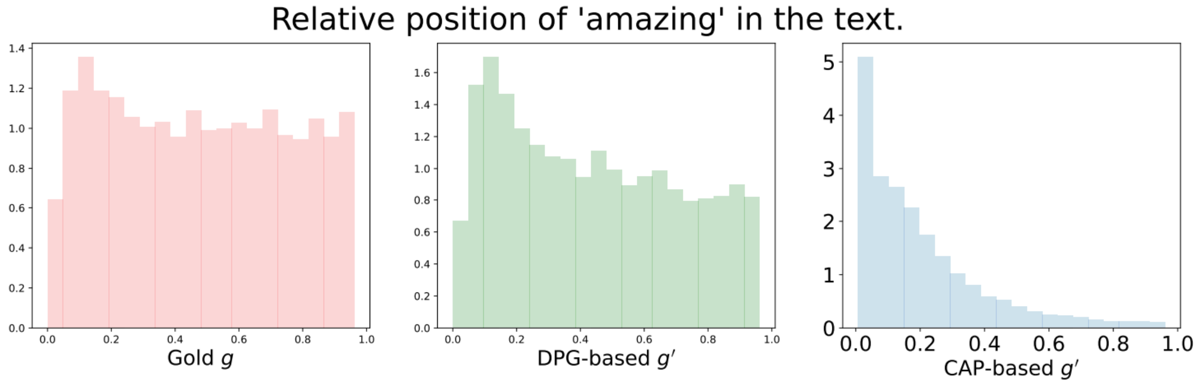


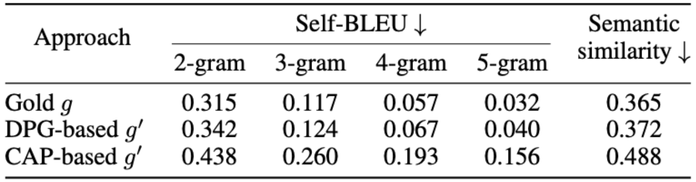


- CAP achieve nice $AR_{a'}$, but really diverges in terms of $KL(g\,|\,g')$.
- Within the same sampling budget, warm-start DPG shows the best Pareto-optimality.
- While preserving a high proximity to g, AR improvement is **180x** (0.0023 -> 0.416) and **60x** (0.005 -> 0.306)

Experimental Results



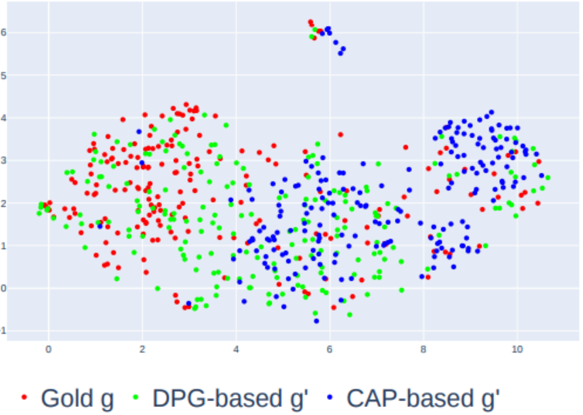

Relative position of 'amazing' in the text.

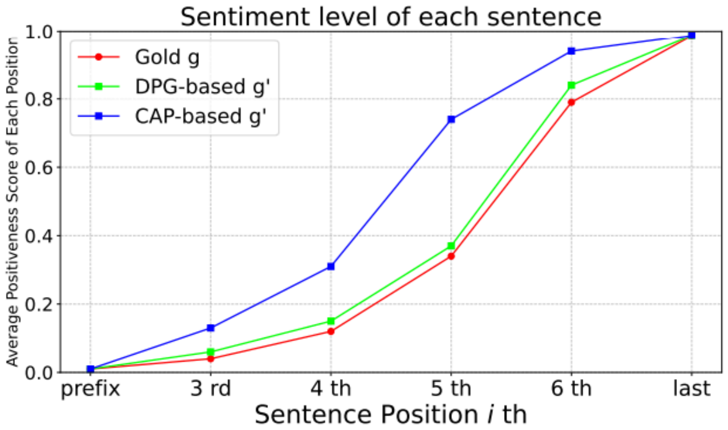


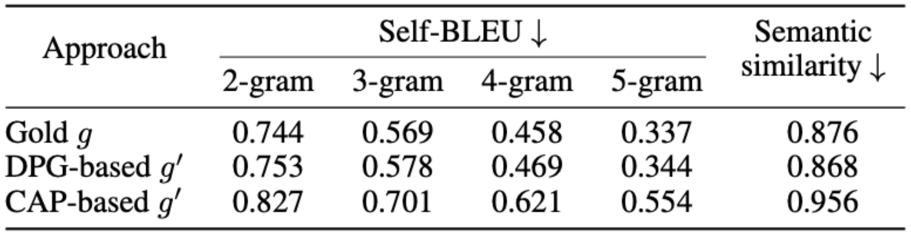
Approach	Self-BLEU ↓				Semantic similarity ↓
	2-gram	3-gram	4-gram	5-gram	
Gold g	0.315	0.117	0.057	0.032	0.365
DPG-based g'	0.342	0.124	0.067	0.040	0.372
CAP-based g'	0.438	0.260	0.193	0.156	0.488

- Also in our analysis, we can detect distributional distortion with CAP: **positional bias** and **loss of diversity**.
- But, GUARD training preserves these properties much better.
- **GUARD naturally satisfies the “keywords” constraints.**

Experimental Results







Approach	Self-BLEU ↓				Semantic similarity ↓
	2-gram	3-gram	4-gram	5-gram	
Gold g	0.744	0.569	0.458	0.337	0.876
DPG-based g'	0.753	0.578	0.469	0.344	0.868
CAP-based g'	0.827	0.701	0.621	0.554	0.956

- Also, in our analysis, we can detect distributional distortion with CAP: **positional bias** and **loss of diversity**.
- But, GUARD training preserves these properties much better.
- **GUARD naturally satisfies the “positive ending” constraints.**

Conclusion

- We obtain 💯 **guarantees** in LLMs while **preserving original distribution** as much as possible.

- First, we formalize what is the optimal distribution g for guaranteed generation.

- And this distribution g cannot be achieved only with training, that is why we need **harmonization between training-time & inference-time methods**.

- We present a **theorem** in the form of an equality that **jointly optimizes** distributional closeness and acceptance rate, with inference-time rejection sampling. And **suggest GUARD baselines: CAP, SFT, and DPG**.

- Experimentally, we confirmed that **warm-start DPG** maintains proximity to the ideal distribution while increasing the acceptance rate by nearly a **100x**, and various analyses support this finding.

Additional Remark

- Two types of tasks: **Relativity** and **Absoluteness**
- Human preference is **relative**, but reasoning, safety, code compilability is **absolutely correct or wrong**.
- But, many previous works do **RL style training for absolute tasks**.

KL-regularized RL's ideal distribution: $g(y|x) \propto a(y|x)\exp(\frac{b(x)}{\beta})$

-> Allocates **non-zero probability** for absolutely wrong output y.
-> β decides how trained model can be diverged from a.

GUARD's ideal distribution: $g(y|x) \propto a(y|x)b(x)$

-> Allocates **zero probability** for absolutely wrong output y.
-> It is the closest distribution from a with perfectly satisfying constraint $b(y)$.

It is why we should **pay attention to GUARD training objective** for solving **absolute** tasks.

Thank you!