Supplementary Material

1. Supplementary material for the "Tumor size, NLR and survival data modeling" section in the main manuscript

1.1. General description of COX model

The Cox proportional hazard model is a regression model used for time-to-event analysis. The structure of this model is based on the assumption that for each individual *i*, the set of Q baseline covariates {Cov_{li} , *l = 1,2,…,Q*} determines individual hazard function $h_i(t)$ and, therefore, individual survival function $S_i(t)$. Mathematically, it is determined by the following equations:

$$\begin{cases} h_i(t) = h_0(t) * \exp(\sum_{l=1}^{Q} \gamma_l * Cov_{li}), \\ \quad S_i(t) = \exp\left(-\int_0^t h_i(t)dt\right), \end{cases} \tag{1}$$

where $h_0(t)$ denotes the baseline hazard function and $\{\gamma_l, l = 1,2,\dots,Q\}$ are association coefficients. Both $h_0(t)$ and γ_l are the parameters of the COX model. Parameter values are obtained using the available survival dataset, in a procedure usually referred to as a qualification of model, which is based on the maximization of a likelihood function. Baseline hazard $h_0(t)$ is often restricted to a class of functions described by a finite set of parameters, for instance, piecewise-constant functions, spline approximations, or Weibull distributions.

1.2. Covariate search procedure using COX models

This research is primarily focused on the use of SLD and NLR, even though we tested up to two additional baseline covariates, to increase the prediction performance of the survival model . Here we describe the covariate search procedure used for the selection of these covariates.

Based on the available clinical data in the training dataset, the following covariates – beyond SLD and NLR – were tested:

- ECOG performance status [s1] (0 or 1), referred to as ECOG;
- PD-L1 expression level [s2] (high, low, or unknown), referred to as PDL1;
- Smoking status (smoker, non-smoker, or ex-smoker), referred to as SMK;
- EGFR status [s3] (positive or negative), referred to as EGFR;
- Patient age at start of treatment (a positive integer), referred to as AGE.

To rank these covariates, we qualified a set of COX models on the training dataset (ATLANTIC patients). All models were built using the *coxph()* function from the *survival* package, version 2.44-1.1, in the R software [s4]. A piecewise-constant approximation was used for the baseline hazard function. Each model included logarithmically transformed SLD, NLR, and one covariate from the list above. The

Akaike Information Criterion (AIC) [s5] and the Likelihood-Ratio Test (LRT) [s6] were estimated for all models.

Based on the AIC and LRT estimates shown in Table S1, we concluded that PDL1 and ECOG provided the highest increase in likelihood in the corresponding COX models *vs.* other covariates. For further investigation of longitudinal SLD and NLR, we chose a COX model with SLD+NLR+PDL1+ECOG as a basic model. For the remainder of this document, this model is simply referred to as “COX”. The association coefficients of this model are presented in Table S2.

1.3. General description of the joint model

The longitudinal joint model (JM) was described as a generalization of the COX model for longitudinal biomarkers. In terms of a hazard function, it features an additional component, $\sum_{k=1}^{P}(\alpha_k * m_{ki}(t))$, which represents the impact of the longitudinal biomarker. Survival is thus formulated as follows:

$$S_i(t) = \exp\left(-\int_0^t h_0(t) * \exp(\textstyle\sum_{l=1}^{Q}(\gamma_l * Cov_{li}) + \sum_{k=1}^{P}(\alpha_k * m_{ki}(t)))\, dt\right), \qquad (2)$$

where α_k is the association parameter vector, a key metric for the biomarker impact on the risk of event, and $m_{ki}(t)$ is an individual vector of longitudinal data (biomarkers), analyzed using a mixed-effects sub-model. The mixed-effects sub-model is an important component of joint modeling; it is used to describe individual variability and to handle stochastic deviations in biomarker measurements. Mixed-effects models may be of a linear or non-linear nature, and various distributions of random effects and types of residual error may be applied. These models are named “joint” because their parameters are estimated using a joint likelihood function, which captures both the time-to-event impact and the longitudinal data likelihood in a mixed-effects sub-model.

1.4. Univariate JM SLD model description

The baseline hazard was parameterized using a Weibull distribution with a transformed scale parameter

$$h_0(t) = p[e^{\lambda}]([e^{\lambda}]t)^{p-1}$$

Survival was thus defined as follows:

$$S_i(t) = \exp\left(-\int_0^t p[e^{\lambda}]\left([e^{\lambda}]t\right)^{p-1} * \exp(\gamma_1\, ECOG_i + \gamma_2\, PDL1_i + \gamma_3\, NLR_i + \alpha\, y_i(t))dt\right)$$

A logarithmic transformation was used for SLD, and a linear-exponential mixed-effects sub-model was formulated as follows:

$$\begin{cases} \ln(SLD_{ij} + 1) = y_i(t_{ij}) + \varepsilon_{ij}, \\ y_i(t_{ij}) = (c * c_i) * \exp[(d + d_i)t_{ij}] + (b * b_i) * t_{ij}, \\ b_i \sim Lognormal(1, \beta_1), \quad c_i \sim Lognormal(1, \beta_2), \quad d_i \sim Normal(0, \beta_3), \\ \varepsilon_{ij} \sim Normal(0, \sigma). \end{cases} \qquad (3)$$

where t_{ij} and SLD_{ij} are timepoints and values of SLD measured for the *i*-th patient. Parameters b > 0, c > 0, and d are fixed effects. Random values $b_i,\ c_i,\ d_i$ are individual random effects described by parameters β_1, β_2 , β_3 of corresponding distributions; ε_{ij} is a normally distributed residual error.

1.5. Multivariate JM SLD&NLR model description

The baseline hazard was parameterized using a Weibull distribution with transformed scale parameter

$$h_0(t) = p[e^{\lambda}]([e^{\lambda}]t)^{p-1}$$

Survival was defined as follows, with a bilinear association structure chosen for the two biomarkers:

$$S_i(t) =$$

$$= \exp\left(-\int_0^t p[e^{\lambda}]([e^{\lambda}]t)^{p-1} * \exp(\gamma_1\ ECOG_i + \gamma_2\ PDL1_i + \alpha_1\ y_i(t) + \alpha_2\ z_i(t) + \alpha_{12}\ y_i(t)\ z_i(t))dt\right).$$

A logarithmic transformation was used for both SLD and NLR. A linear-exponential mixed-effects sub-model was applied to SLD, while a hyperbolic sub-model was used to describe longitudinal NLR:

$$\begin{cases} \ln(SLD_{ij}+1) = y_i(t_{ij}) + \varepsilon_{ij},\ \ \ln(NLR_{ik}+1) = z_i(\tau_{ik}) + \epsilon_{ik} \\ y_i(t_{ij}) = (c * c_i) * \exp\left[(d + d_i)t_{ij}\right] + (b * b_i) * t_{ij}, \\ z_i(\tau_{ik}) = (p + p_i) + (\exp(l + l_i)) * \frac{[(q+q_i)-(p+p_i)]}{\tau_{ik}+\exp(l+l_i)}, \\ b_i \sim Lognormal(1,\beta_1),\quad c_i \sim Lognormal(1,\beta_2),\quad d_i \sim Normal(0,\beta_3), \\ q_i \sim Normal(0,\beta_4),\quad p_i \sim Normal(0,\beta_5),\quad l_i \sim Normal(0,\beta_6) \\ \varepsilon_{ij} \sim Normal(0,\sigma_1),\ \ \epsilon_{ik} \sim Normal(0,\sigma_2). \end{cases} \qquad (4)$$

where t_{ij} and SLD_{ij} are timepoints and values of SLD measured for the *i*-th patient; τ_{ik} and NLR_{ik} are timepoints and values of NLR. Parameters b > 0, c > 0, and d are fixed-effects for SLD; q, p and l are fixed effects for NLR. Random effects $b_i,\ c_i,\ d_i, q_i,\ p_i,\ l_i$ are described by parameters β_1, β_2 , β_3, β_4, β_5 , β_6 of corresponding distributions, and ε_{ij}, ϵ_{ik} are normally distributed residual errors.

1.6. Qualification of models

The likelihood composition for joint models was based on the assumption of conditional independence of all longitudinal measurements SLD_{ij} and NLR_{ik} (for multivariate JM). Fixed effects, random effects and residuals parameters were also assumed to be statistically independent. For details on the likelihood composition for longitudinal JM, see [s7] and the cited literature therein. To calculate the survival impact on the likelihood, a tanh-sinh quadrature for numerical integration [s8] was used.

To optimize parameters, a Markov Chain Monte Carlo (MCMC) algorithm was used for sampling from the log-likelihood distribution. It was implemented using the *Stan* software [s9], which offers a Hamiltonian Monte Carlo [s10] sampling method - one of the most efficient MCMC algorithms.

Although a Bayesian approach was used in the estimation of model parameters, priors played a minor role in the final estimates. For positive fixed-effects parameters, log-normal distributions with means of 1 and standard deviations of 1 were selected as priors; for all other parameters, standard normal distributions were chosen as priors (and truncated for positive parameters). Since the training dataset could be described as a rich data sample (200 events, 1507 measurements of SLD and 5055 measurements of NLR), the selected priors should be considered as non-informative.

For both JM SLD and JM SLD&NLR model qualification, a similar sampling setting was applied. Four Markov Chains were used for sampling. In order to ensure successful convergence, each chain featured the following setting: total number of iterations – 4000, from which 2000 were warm-up iterations; initial values for all parameters were set randomly, from a uniform distribution in the [-0.1, 0.1] range (and truncated for positive parameters); parameters for the No-U-Turn sampler in *Stan* - adapt_delta = 0.85, max_treedepth = 12. A summary of the sampling results is presented in Table S3, with columns featuring: *name* – name of the parameter from formulas (3) or (4); *mean* – mean value of sampled results; *se_mean* – Monte Carlo standard error; *sd* - standard deviation of a parameter; *2.5%* and *97.5%* - corresponding quantiles of sampled parameter; *n_eff* - effective sample size; *Rhat* - potential scale reduction factor (MCMC convergence statistics).

We do not have rights to publish the clinical datasets used in this research, and the R scripts we used have been adapted to the format of these particular datasets (the format is generally similar to the one used in the R *JM* package by D. Rizopoulos). We share, however, the most important components of this modeling framework by listing the *Stan* code for the multivariate JM SLD&NLR model, in Appendix A of these Supplementary Materials. For full details on model development and evaluation, please do not hesitate to contact the corresponding author.

2. Supplementary material for the "Survival predictions for patient subgroups in validation study" section in the main manuscript

2.1. Individual survival estimates using JM

Provided the following individual patient dataset - set of baseline covariates values ω_j and longitudinal information collected up to time T^s $\{\mathcal{Y}_i(s_i), s_i \leq T^s\}$, let $\mathcal{D}$ denote all parameters of the qualified joint model (JM). The conditional survival probability estimate for selected patients at time $T^h > T^s$ is defined by:

$$\pi(T^h \mid T^s) = \Pr(T^* \geq T^h \mid T^* > T^s, \mathcal{Y}_i, \omega_j, \mathcal{D}) \qquad (5)$$

where T^* denotes the event time.

Survival is computed in *Stan,* using a Markov chain Monte Carlo algorithm for sampling from the conditional distribution (5). It is implemented using *Stan*. The mean value of $\pi(T^h \mid T^s)$ is used as a final estimate for further ROC-AUC and BS calculations, as well as for other validation scenarios.

3. Supplementary material for the “Precision of individual survival predictions” section in the main manuscript

3.1. ROC-AUC and BS calculation method

Since both training and validation datasets are characterized by a high amount of right-censored patients, an Inverse Probability of Censoring Weighting (IPCW) approach was used to estimate ROC-AUC and BS [s11]. Let $\hat{G}_{T^s}(t)$ denote the Kaplan-Meier estimator for $\Pr(C > t \mid C > T^s)$, where C is a time of censoring, calculated on a subset of patients DS_{T^s} known to be in the study up to time T^s. $ROC-AUC(T^h,T^s)$ is then defined as:

$$ROC-AUC(T^h,T^s) = \\ = \frac{\sum_{i=1}^{n}\sum_{j=1}^{n} \mathrm{I}_{(T_i^* \le T^h, \nu_i^*=1)} 1/\hat{G}_{T^s}(T_i^*) \mathrm{I}_{(T_j^* > T^h)} 1/\hat{G}_{T^s}(T^h) \mathrm{I}_{(\pi_i(T^h \mid T^s) > \pi_j(T^h \mid T^s))}}{(\sum_{i=1}^{n} \mathrm{I}_{(T_i^* \le T^h, \nu_i^*=1)} 1/\hat{G}_{T^s}(T_i^*))(\sum_{j=1}^{n} \mathrm{I}_{(T_j^* > T^h)} 1/\hat{G}_{T^s}(T^h))},$$

where n is the total number of patients in DS_{T^s}, T_i^* is an observed event time for i patient, ν_i^* denotes the type of an event: a value of 1 specifies the true event took place (patient died), a value of 0 specifies the patient was censored.

BS was defined as follows:

$$BS(T^h,T^s) = 1/n(\sum_{i=1}^{n} \mathrm{I}_{(T_i^* \le T^h, \nu_i^*=1)} (\pi_i(T^h \mid T^s))^2/\hat{G}_{T^s}(T_i^*) + \\ + \sum_{j=1}^{n} \mathrm{I}_{(T_j^* > T^h)} (\pi_i(T^h \mid T^s) - 1)^2/\hat{G}_{T^s}(T^h)).$$

ROC-AUC and BS were estimated using a standard R functionality.

4. Supplementary material for the “Results” section in the main manuscript

All three models (COX, JM SLD, JM SLD&NLR) were qualified using the training dataset, *i.e.* patients from the ATLANTIC clinical study. In the main text of the paper, we presented excerpts of ROC-AUC and BS diagnostics, namely, metrics estimated for different cut-offs T^s of longitudinal data calculated for survival predictions in the validation dataset (patients from the 1108 clinical study), at time of prediction T^h=12 months after the start of therapy. Here we provide complete ROC-AUC and BS diagnostics, which have been performed for both training and validation datasets.

4.1. Internal ROC-AUC and BS validation

Based on the training dataset and using different amounts of longitudinal data from T^s =0 to T^s=6 months, we calculated ROC-AUC and BS at each month up to T^h=24 months following the start of treatment. Figure S1 displays the calculated metrics for T^s= 3 months graphically. A summary for T^h= 12 months and all considered T^s is presented in Table S4.

4.2. External ROC-AUC and BS validation

The external validation was performed similarly to the internal validation procedure (Section 4.1). ROC-AUCs and BSs were calculated for different T^s from 0 to 6 months, at each month, up to T^h= 24 months following the start of treatment. Figure S2 features the calculated metrics for T^s= 3 months graphically. A summary for T^h=12 months and all considered T^s is presented in Table S4.

4.3. Validation of longitudinal biomarkers; description and predictions

In order to show how well selected mixed-effects sub-models for SLD and NLR describe these biomarkers, we performed Visual Predictive Check (VPC) diagnostics for the multivariate JM SLD&NLR model. Figures S3 (a) and (b) represent VPC diagnostics and summarize the distribution of observed biomarker values in the training dataset vs. predicted values during model qualification, for SLD and NLR. Figures S3 (c) and (d), similarly to Figures S3 (a) and (b), summarize the distribution of observed biomarkers in the validation dataset vs. predicted values obtained from the posterior distribution of the qualified JM SLD&NLR model informed with longitudinal data from the validation dataset. Figures S3 (e) and (f) extend the VPC diagnostics and show how well this model predicted SLD and NLR longitudinal dynamics in the validation dataset, based on a longitudinal data cut-off of T^s = 3 months.

References

[s1] Oken MM, Creech RH, Tormey DC, et al. Toxicity and response criteria of the Eastern Cooperative Oncology Group. *Am. J. Clin. Oncol*. 1982;5(6): 649–55. doi:10.1097/00000421-198212000-00014

[s2] Garassino MC, et al. Durvalumab as third-line or later treatment for advanced non-small-cell lung cancer (ATLANTIC): an open-label, single-arm, phase 2 study. *Lancet Oncol*. 2018 Apr;19(4):521-536. doi: 10.1016/S1470-2045(18)30144-X.

[s3] Bethune G, Bethune D, Ridgway N, Xu Z. Epidermal growth factor receptor (EGFR) in lung cancer: an overview and update. *J Thorac Dis*. 2010;2(1):48–51.

[s4] Survival package for R. https://github.com/therneau/survival.

[s5] Akaike H. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*. 1974; 19(6):716-723.

[s6] Glover S, Dixon P. Likelihood ratios: A simple and flexible statistic for empirical psychologists. *Psychonomic Bulletin & Review*. 2004; 11:791–806. https://doi.org/10.3758/BF03196706

[s7] Brilleman SL et al. Joint longitudinal and time-to-event models for multilevel hierarchical data. *Stat Methods Med Res*. 2019 Dec;28(12):3502-3515

[s8] Takahasi H, Mori M. Double Exponential Formulas for Numerical Integration, *Publications of the Research Institute for Mathematical Sciences*. 1974;9 (3): 721–741

[s9] Carpenter B, Gelman A, Hoffman M. et al. Stan: A Probabilistic Programming Language. *Journal of Statistical Software*. 76 (1): 1–32. doi:10.18637/jss.v076.i01

[s10] Duane S, Kennedy AD, Pendleton BJ, Roweth D. Hybrid Monte Carlo. *Physics letters B*. 1987;195(2):216–222, 1987.

[s11] Blanche P, Latouche A, Viallon V. “Time-Dependent AUC with Right-Censored Data: A Survey” in *Risk Assessment and Evaluation of Predictions*, New York, Springer, 2013.

Appendix A.

We here provide the *Stan* code used for the qualification of the multivariate JM SLD&NLR model. This listing contains two scripts: the content of the *.stan* file and the content of the additional *.hpp* file, which includes the implementation of computationally demanding likelihood calculations in C++.

```
// content of .stan file
// multivariate nonlinear joint model JM SLD&NLR
functions {
  real ll(int n, real[] fixranef, real[] params, real[] event_times, int[] events,
  int threads); // log-likelihood impact of time-to-event data implemented in c++
}
data {
  int<lower=0> n; // number of patients
  int<lower=0,upper=1> events[n]; // events
  real<lower=0> event_times[n]; // event times
  int<lower=0> y1n; // length of y1 (SLD)
  int<lower=0> y2n; // length of y2 (NLR)
  real y1[y1n]; // SLD measurements
  real y2[y2n]; // NLR measurements
  real t1[y1n]; // SLD measurements timepoints
  real t2[y2n]; // NLR measurements timepoints
  int<lower=0> y1_l[n]; // indexes y1
  int<lower=0> y2_l[n]; // indexes y2
  int<lower=0> factor1_level_n; // levels length of factor1 (ECOG)
  int<lower=1> factor1[n]; // indexes of factor1 (ECOG)
  int<lower=0> factor2_level_n; // levels length of factor2 (PDL1)
  int<lower=1> factor2[n]; // indexes of factor2 (PDL1)
  int<lower=1> threads;
  // priors
  real<lower=0> bs_haz_p_prior_sd;
  real<lower=0> bs_haz_lambda_prior_sd;
  real<lower=0> bs_haz_p_prior_intercept;
  real bs_haz_lambda_prior_intercept;
  real<lower=0> assoc_coefs_prior_sd[3];
  real<lower=0> factor_coefs_prior_sd;
  real assoc_coefs_prior_intercept[3];
  real<lower=0> y1_prior_sd[7]; // [1:3] - fixed eff, [4] - sigma [5:7] - rand eff,
  real<lower=0> y2_prior_sd[7]; // [1:3] - fixed eff, [4] - sigma [5:7] - rand eff
  real y1_prior_intercept[3]; // [1:3] - fixed eff
  real y2_prior_intercept[3]; // [1:3] - fixed eff
}
parameters {
  real<lower=0> bs_haz_p; // p
  real bs_haz_lambda_e; // lambda
  real assoc_coefs[3]; // [1] - alpha_1 (SLD), [2] - alpha_2 (NLR), [3] - alpha_{12}
  real factor1_coefs[factor1_level_n-1]; // gammas_1
  real factor2_coefs[factor2_level_n-1]; // gammas_2
  real<lower=0> y1_fixed_b; // b (SLD)
  real<lower=0> y1_fixed_c; // c (SLD)
  real y1_fixed_d; // d (SLD)
  real y2_fixed_b; // q (NLR)
  real y2_fixed_c; // p (NLR)
  real y2_fixed_l; // l (NLR)
  real<lower=0> y1_ranef[3]; // [1] - beta_1, [2] = beta_2, [3] - beta_3
  real<lower=0> y2_ranef[3]; // [1] - beta_4, [2] = beta_5, [3] - beta_6
  real<lower=0> sigma1;
  real<lower=0> sigma2;
  real<lower=0> ranef1_b[n]; // b_i (SLD)
  real<lower=0> ranef1_c[n]; // c_i (SLD)
  real ranef1_d[n]; // d_i (SLD)
  real ranef2_b[n]; // q_i (NLR)
  real ranef2_c[n]; // p_i (NLR)
  real ranef2_l[n]; // l_i (NLR)
}
transformed parameters {
    real bs_haz_lambda;
  real fixranef[7*n];
  bs_haz_lambda = exp(bs_haz_lambda_e); // transformed lamda
  for (i in 1:n) {
    int ind = (i-1)*7;
    // calculate parameters from fixed and random effects
    fixranef[ind+1] = ranef1_b[i] * y1_fixed_b;
    fixranef[ind+2] = ranef1_c[i] * y1_fixed_c;
    fixranef[ind+3] = ranef1_d[i] + y1_fixed_d;
    fixranef[ind+4] = ranef2_b[i] + y2_fixed_b;
    fixranef[ind+5] = ranef2_c[i] + y2_fixed_c;
    fixranef[ind+6] = exp(ranef2_l[i] + y2_fixed_l);
    // calculate factor-type covariates
```

```
      if (factor1[i] == 1) fixranef[ind+7] = 0;
      else fixranef[ind+7] = factor1_coefs[(factor1[i]-1)];
      if (factor2[i] == 1) fixranef[ind+7] += 0;
      else fixranef[ind+7] += factor2_coefs[(factor2[i]-1)];
    }

}
model {
  // main ll
  real params[6];
  real lol = 0;
  int st;
  int en;
  int ind;
  real delta;
  real v;
  params[1] = bs_haz_p;
  params[2] = bs_haz_lambda;
  params[3:4] = assoc_coefs[1:2];
  params[5] = assoc_coefs[3];
  params[6] = 0.330;

  // random effects
  lol += lognormal_lpdf(ranef1_b | 1, y1_ranef[1]);
  lol += lognormal_lpdf(ranef1_c | 1, y1_ranef[2]);
  lol += normal_lpdf(ranef1_d | 0, y1_ranef[3]);
  lol += normal_lpdf(ranef2_b | 0, y2_ranef[1]);
  lol += normal_lpdf(ranef2_c | 0, y2_ranef[2]);
  lol += normal_lpdf(ranef2_l | 0, y2_ranef[3]);

  // time-to-event
  lol += ll(n, fixranef, params, event_times, events, threads);


  // y1 (SLD) impact
  st = 1;
  for (i in 1:n) {
    en = y1_l[i];
    ind = (i-1)*7;
    for (j in st:en) {
      v = fixranef[ind+2]*exp(fixranef[ind+3]*t1[j]) + t1[j]*fixranef[ind+1];
      lol += normal_lpdf(v | y1[j], sigma1);
    }
    st = en+1;
  }

  // y2 (NLR) impact
  st = 1;
  for (i in 1:n) {
    en = y2_l[i];
    ind = (i-1)*7;
    for (j in st:en) {
      v = fixranef[ind+5] + fixranef[ind+6]*(fixranef[ind+4] -
      fixranef[ind+5])/(t2[j]+fixranef[ind+6]);
      lol += normal_lpdf(v | y2[j], sigma2);
    }
    st = en+1;
  }


  // apply priors
  lol += normal_lpdf(bs_haz_p | bs_haz_p_prior_intercept, bs_haz_p_prior_sd);
  lol += normal_lpdf(bs_haz_lambda_e | bs_haz_lambda_prior_intercept,
  bs_haz_lambda_prior_sd);
  lol += normal_lpdf(assoc_coefs | assoc_coefs_prior_intercept, assoc_coefs_prior_sd);
  lol += normal_lpdf(y1_ranef | 0, y1_prior_sd[5:7]);
  lol += normal_lpdf(y2_ranef | 0, y2_prior_sd[5:7]);
  lol += lognormal_lpdf(y1_fixed_b | y1_prior_intercept[1], y1_prior_sd[1]);
  lol += lognormal_lpdf(y1_fixed_c | y1_prior_intercept[2], y1_prior_sd[2]);
  lol += normal_lpdf(y1_fixed_d | y1_prior_intercept[3], y1_prior_sd[3]);
  lol += normal_lpdf(y2_fixed_b | y2_prior_intercept[1], y2_prior_sd[1]);
  lol += normal_lpdf(y2_fixed_c | y2_prior_intercept[2], y2_prior_sd[2]);
  lol += normal_lpdf(y2_fixed_l | y2_prior_intercept[3], y2_prior_sd[3]);
  lol += normal_lpdf(sigma1 | 0, y1_prior_sd[4]);
  lol += normal_lpdf(sigma2 | 0, y2_prior_sd[4]);
  target += normal_lpdf(factor1_coefs | 0, factor_coefs_prior_sd);
  target += normal_lpdf(factor2_coefs | 0, factor_coefs_prior_sd);


  target += lol;
}
```

```
//
//
//

// content of .hpp file with implementation of ll() function

//
//
//


# define M_PI 3.14159265358979323846  /* pi */

const double nodes_tanhsinh[] = {
    /* 1st layer nodes: transformed 0, 1, 2, 3 */
    0.00000000000000000000,
    0.95136796407274694573,
    0.99997747719246159286,
    0.99999999999995705839,
    /* 2nd layer nodes: transformed 1/2, 3/2, 5/2 */
    0.67427149224843582608,
    0.99751485645722438683,
    0.99999998887566488198,
    /* 3rd layer nodes: transformed 1/4, 3/4, ... */
    0.37720973816403417379,
    0.85956905868989663517,
    0.98704056050737689169,
    0.99968826402835320905,
    0.99999920473711471266,
    0.99999999995285644818,
    /* 4th layer nodes: transformed 1/8, 3/8, ... */
    0.19435700332493543161,
    0.53914670538796776905,
    0.78060743898320029925,
    0.91487926326457461091,
    0.97396686819567744856,
    0.99405550663140214329,
    0.99906519645578584642,
    0.99990938469514399984,
    0.99999531604122052843,
    0.99999989278161241838,
    0.99999999914270509218,
    0.99999999999823216531
  };

const double weights_tanhsinh[] = {
    /* First layer weights */
    1.5707963267948966192,
    0.230022394514788685,
    0.00026620051375271690866,
    1.3581784274539090834e-12,
    /* 2nd layer weights */
    0.96597657941230114801,
    0.018343166989927842087,
    2.1431204556943039358e-7,
    /* 3rd layer weights */
    1.3896147592472563229,
    0.53107827542805397476,
    0.076385743570832304188,
    0.0029025177479013135936,
    0.000011983701363170720047,
    1.1631165814255782766e-9,
    /* 4th layer weights */
    1.5232837186347052132,
    1.1934630258491569639,
    0.73743784836154784136,
    0.36046141846934367417,
    0.13742210773316772341,
    0.039175005493600779072,
    0.0077426010260642407123,
    0.00094994680428346871691,
    0.000062482559240744082891,
    1.8263320593710659699e-6,
  1.8687282268736410132e-8,
  4.9378538776631926964e-11
 };

const int offsets_tanhsinh[] = { 1, 4, 7, 13, 25 };

template<typename T>
inline T y_t_explin(const T& t, const T& b, const T& c, const T& d) {return c*exp(d*t) + b*t;}

template<typename T>
inline T y_t_bcl(const T& t, const T& b, const T& c, const T& l) {return c + l*(b-c)/(t+l);}

template<typename T>
inline T h0_t_wei(const T& t, const T& p, const T& lambda) {return p*lambda*pow(lambda*t, p-1.0);}

template<typename T1, class T2>
T1 tanhsinh(const T1& a, const T1& b, int lmax, const T2& y, const std::vector<T1>& y_p, T1 eps = T1(1.0e-10)) {

 T1 p = (a + b) * 0.5;
 T1 q = (b - a) * 0.5;

 if (abs(q) < eps) return T1(0);

 T1 integral = T1(weights_tanhsinh[0]) * y(p, y_p);
 int m = (2 < lmax) ? lmax : 2;
 m = (4 < lmax) ? 4 : lmax;

 for (int i = offsets_tanhsinh[0]; i<offsets_tanhsinh[m]; ++i) {
  integral += T1(weights_tanhsinh[i]) *
   (y(p + q * T1(nodes_tanhsinh[i]), y_p) + y(p - q * T1(nodes_tanhsinh[i]), y_p));
 }
 integral *= q*pow(0.5, m-1);

 return integral;
}

template<typename T>
T h_t(const T& t, const std::vector<T>& p) {
  // p[0], p[1] - weibull params
  T h = h0_t_wei(t, p[0], p[1]);
  // next 3 params are related to the 1st biomarker
  T y1 = y_t_explin(t, p[2], p[3], p[4]);
  // next 3 params are related to the 2nd biomarker
  T y2 = y_t_bcl(t, p[5], p[6], p[7]);
  // p[8] = koef_y1, p[9] = koef_y2, p[10] = koef_y1y2, p[11] = factor impact
  h *= exp(p[8]*y1 + p[9]*y2 + p[10]*y1*y2 + p[11]);
  return h;
}


template<typename T>
T s_t(const T& t, const std::vector<T>& p, const T& maxtime) {
 if (t < maxtime) return tanhsinh(T(0), t, 2, h_t<T>, p);
 else return tanhsinh(T(0), t, 3, h_t<T>, p);
}


template <typename T1__, typename T2__, typename T3__>
typename boost::math::tools::promote_args<T1__, T2__, T3__>::type
ll(const int& n, // n - number of patients
     const std::vector<T1__>& fixranef, // array of individual parameters length=n*6
     const std::vector<T2__>& params, // array of params length=6
     const std::vector<T3__>& event_times,
     const std::vector<int>& events,
     const int& threads, //
     std::ostream* pstream__) {

  typedef typename boost::math::tools::promote_args<T1__, T2__, T3__>::type
```

```
      result_type;

    result_type logl = 0;

      // global cycle for each id

    std::vector<result_type> p_i(12);
    p_i[0] = result_type(params[0]); // p
    p_i[1] = result_type(params[1]); // lambda
    p_i[8] = result_type(params[2]); // assoc 1
    p_i[9] = result_type(params[3]); // assoc 2
    p_i[10] = result_type(params[4]); // assoc 12
    result_type maxtime = result_type(params[5]); // time to ease on integration

    for(int i=0; i<n; ++i){

      p_i[2] = result_type(fixranef[7*i + 0]); // y1
      p_i[3] = result_type(fixranef[7*i + 1]); // y1
      p_i[4] = result_type(fixranef[7*i + 2]); // y1
      p_i[5] = result_type(fixranef[7*i + 3]); // y2
      p_i[6] = result_type(fixranef[7*i + 4]); // y2
      p_i[7] = result_type(fixranef[7*i + 5]); // y2
      p_i[11] = result_type(fixranef[7*i + 6]); // factor hazard

        logl += -s_t(result_type(event_times[i]), p_i, maxtime); // log(S(t)) =
        int_0^t{h(x)dx}
      if (events[i]==1) {logl += log(h_t(result_type(event_times[i]), p_i));} // +
      log(h(t))
    }

      return logl;
  }
```